**OPAALS PROJECT**

Contract n° IST-034824

# WP5: Integration with the Digital Ecosystem Platform

# DELIVERABLE 5.9

# Test Plan. Evaluation report and recommendations from Beta-testing of P2P infrastructure

Project funded by the European Community under the "Information Society Technology" Programme

**Project Acronym**: OPAALS

**Deliverable N°**: 5.9

**Due date**: May 30st 2010

**Delivery Date**: July 1st 2010

**Short Description** This Deliverable presents how to validate a platform based on a p2p distributed network. To accomplish this goal is realized, first elicit requirements, and continue with the selection of the most important requirements, performing a state of the art test tools p2p environments, and finally verification of the p2p framework.

**Authors** ITA: F.J. Lacueva, Jorge I. Vea-Murguía, Pilar Fernández de Alarcón, Juanjo Navamuel.

**Partners contributed**: Instituto Tecnológico de Aragón (ITA),

**Made available to**: July, 1st 2010

| Versioning | | |
|---|---|---|
| **Version** | **Date** | **Name, organization** |
| 0.1 | 21th September 2009 | F.J. Lacueva & J. Vea-Murguía (ITA). |
| 0.2 | 15th September 2009 | F.J. Lacueva (ITA). |
| 0.4 | 12th December 2009 | P. Fernández de Alarcón (ITA) |
| 0.5 | 15th June 2010 | Juanjo Navamuel (ITA) |
| 0.6 | 30th June 2010 | J. Vea-Murguía (ITA). |
| 0.7 | 1st July 2010 | J. Vea-Murguía; FJLacueva (ITA). |
| 0.8 | 15th July 2010 | J.Finnegan (WIT); P. Fernández de Alarcón (ITA); J.Vea-Murguia. |
| 1.0 | 2nd August 2010 | F.J. Lacueva (ITA) |

**Quality check**

**Internal Reviewers**: Paul Krause (Surrey), Jason Finnegan (WIT)

## Dependencies:

| | |
|---|---|
| **Achievements*** | The following research activities and tasks underlying this Deliverable were **completed**: (1) Requirements must have a distribute platform ;(2) *State of the art in distributed test* ;(3)list of test, and beta-test in Opaals p2p platform, Flypeer; |
| **Work Packages** | **WP5** (Integration with the Digital Ecosystem Platform) this deliverable describes the tests ITA performs to validate the requirements of the DE platform (flypeer). |
| **Partners** | London School of Economics, Universität Kassel, Waterford Institute of Technology |
| **Domains** | P2P platforms, Flypeer |
| **Targets** | DSC and DE researchers, SMEs in the IST branch |
| **Publications*** | The reported work and applications source code was published on the SourceForge website at: http://opaalstools.sourceforge.net |
| **PhD Students*** | F. J. Lacueva |
| **Outstanding features*** | (1) Make a state of art of tools to test framework peer 2 peer and make develop a tools to test the framework inside a real networks.<br>(2) Selected the requirements to test the framework<br>(3) Test the all delivery of the project Flypeer, documentation and software.<br>(4) Selected requirement test against the framework |
| **Disciplinary domains of authors*** | ITA: F. J Lacueva, J Vea-Murguía (Software Engineers), P. Fernández de Alarcón, J. Navamuel (Telecommunications Engineers). |

*The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report*

# Table of contents

# Figures and Tables

# 1 EXECUTIVE SUMMARY

The Objective of this paper is to Verify and Validate the proper Functioning of the platform. Verification and Validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It is normally part of the software testing process of a project.

To validate a product it is necessary that an explicit set of requirements are satisfied. In engineering, a requirement is a singular documented need of a particular product. It is most commonly used in a formal sense in systems engineering or software engineering, to identify the characteristics of a product. Requirements are usually written to talk about the same product or feature between the different persons that they are working in a project. This means that the requirements must be unambiguous, complete, consistent, traceable, verifiable, clearly, easy to understand both for normal users and for developers.

Taking into account that none previous requirements document of the project has been chosen, this V&V process has been done considering requirements implemented in a system with similar characteristics. All conclusions have been drawn on some general requirements which are not explicitly given either by developers or by other person involves in the project.

The most important component of the implementation is the way how to connect the different nodes of the application. A study was carried out on a JXTA network and its most important features. Flypeer has been selected as the framework for transaction model tested on this project and results can be found in this deliverable. Flypeer tries to provide a fully distributed environment, which executes different type of order service compositions.

For V&V requirements of the system, the most suitable tool to perform the test of the application should be chosen. This point is not trivial, for it has conducted a study to select the most appropriate tool or to make a small application that does not involve a great effort to run the test every time you make a change in the application.

Manual tests were carried out on different versions for verification, as it has been published. Lists of problems have to be found and shown. The main weakness found in the entire application is the lack of clear documentation related to application requirements needed for evaluation and developers who are the users of the application. The proposed tests are hard, despite that fact the application has passed many of them. In conclusion we will say that tests are inconclusive and not definitive as the application has been validated against general requirements.

# 2 INTRODUCTION

The objective of this deliverable is to sum up all test results and recommendations for future implementations considering general p2p infrastructure requirements that were adopted as the requirements for the new implementation of the p2p infrastructure supporting the Guigoh OKS.

As is it explained in Section 4 and more specific in chapter 4.3 the main objective is to evaluate the p2p infrastructure against the requirements defined in the D5.3 *Re-engineered Servent* document.

***Related Tasks.***
In order to establish the task environment we show the related task in the next table.

| Task | Description |
|---|---|
| **T5.8:** | Locking mechanism on the coauthoring tool (Surrey, IPTI) |
| **T5.9:** | P2P infrastructure implementation (IPTI, Surrey) |
| **T5.10** | Alpha-test of P2P infrastructure implementation (SUAS) |
| **T5.11** | Integration of distributed accountability, identity and trust (WIT) |
| **T5.12** | Beta-test of P2P infrastructure implementation (ITA, Surrey). |
| **T 5.13** | DE business case in the SW development sector of Aragon region (ITA) |

We as beta tester are in dependent on all the preceding tasks (from 5.8 to 5.11) and are going to perform black box tests of the P2P infrastructure.

## 2.1 *Software Release Life Cycle.*

A **software release** is the distribution (whether public or private) of an initial or upgraded version of a computer software product. And the software release life cycle is composed of different stages that describe the stability of a piece of software and the amount as the development process proceeds.
Next picture summarize the software development stage from the testing point of view, they are briefly described in next paragraphs.

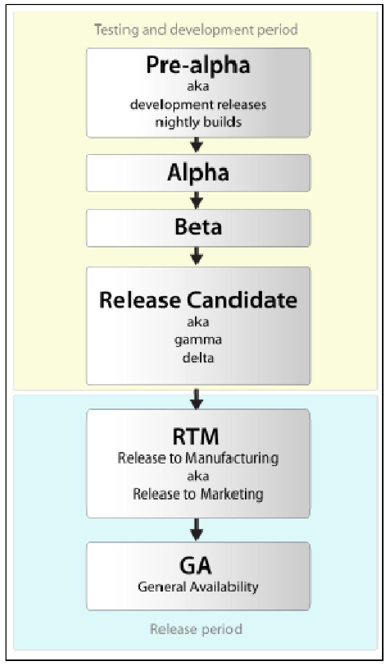



**Table 1.- Software Development and Release Stages.**

- **Pre-Alpha:** In contrast to alpha and beta versions, the pre-alpha is not feature complete. When it is used, it refers to all activities as requirements analysis, software design, software development and unit testing.

- **Alpha:** The alpha build of the software is the build for the internal software testers, that is, people different from the software engineers, sometimes for the public, but usually internal to the organization or community that develops the software. In a rush to market, more and more companies are engaging external customers or value-chain partners in their alpha testing phase. This allows more extensive usability testing during the alpha phase.
In software testing terminology alpha testing is done by the client in the presence of the tester or developers and the test environment is not open for the end user.

- **Beta:** is a nickname for software which has passed the alpha testing stage of development and has been released to users for software testing before its official release. It allows the software to undergo usability testing with users who provide feedback, so that any malfunctions these users find in the software can be reported to the developers and fixed.

- **Release candidate (RC):** refers to a version with potential to be a final product, ready to release unless fatal bugs emerge. In this stage of product stabilization (read QA cycle), all product features have been designed, coded and tested through one or more Beta cycles with no known showstopper-class bug. A release is called code complete when the development team agrees that no entirely new source code will be added to this release.

- **<u>Release to manufacturing or Release to marketing(RTM):</u>** is used to indicate that the software has met a defined quality level and is ready for mass distribution either by electronic means or by physical media.

- **<u>General availability (GA):</u>** is the point where all necessary commercialization activities have been completed and the software has been made available to the general market either via the web or physical media. It is also at this stage that the software is considered to have "gone live". The production, live version is the final version of a particular product. A live release is considered to be very stable and relatively bug-free with a quality suitable for wide distribution and use by end users. In commercial software releases, this version may also be signed (used to allow end-users to verify that code has not been modified since the release). The expression that a software product "has gone live" means that the code has been completed and is ready for distribution. Other terms for the live version include: live master, live release, or live build.

## 2.2 *Scope*

This document has three main goals. The first one is to define a set of tests that can be performed in order to guarantee the correct functioning of a generic P2P multiplatform infrastructure. Once the set is established, a testing tool for automatic deploys, configuration and execution of the test is presented, together with some examples of the scripts that can be used to perform the previously defined tests, which were created to test the Flypeer platform.

And finally, the result of the different tests we were able to perform in order to verify the Flypeer platform were presented (the details are moved to the Annex of this Deliverable) which, as an evaluation and suggestion report is the third goal of this document.

## 2.3 *Document Structured.*

The document is structured as follows:

- The methodology for conducting tests to the Framework Flypeer, How collaborate the test team with the develop team.
- The requirements for p2p applications collected from other project documents. And requirement has to be is selected.
- State of the art test application to distributed applications test. And Speak about own test application to check distribute application.
- Construction of a laboratory test to distribute application, according to the topology p2p, for the verification the Flypeer framework.
- Verification of the framework following the roadmap and documentation on the web

# 3 METHODOLOGICAL REMARKS.

We now describe the working methodology we followed in order to define the test, select the testing tool and perform the test against Flypeer together with the communication we established with the development team in order to raise errors and to be informed of the bugs solution:

- A state of the art is made to test a P2p infrastructure tools and benchmarking algorithms.
- A list of requirement is made to test in the framework to validate.
- Explain the test process:
    - The way in which the test are aligned together with the software releases.
    - The way in which the new releases of Flypeer are communicated to the group together with the documentation about the release features: new functionalities, fixed bugs. To verify the framework.
    - The way in which the tests are planned: a commitment between the development team and the test team.
    - First the development team establishes when a new release is being developed considering the objective goals.
    - When the new version of the Flypeer is released the development team should communicate the state of the developed functionalities: finished, not yet, not started. A new release to be considered finished should contain the complete documentation to deploy the infrastructure and it should allow the execution of the example and agreed tests.
    - As soon as the tests are executed the results should be communicated to the development team. A summary table will be fulfilled:
    - If the test finished successfully the test id together with the ok
    - If some bug were detected the test id together with the bug id.
    - The table should be completed with a detailed description of the bug reported (as far as possible)
    - The development team will complete the summary table with the intended solution: with a new release, some component new release.
    - The documentation of each of the iterations will be included in this document (probably a summary and the details as attachments).

# 4 REQUIREMENTS AND TESTS FOR P2P PLATFORM VALIDATION.

## 4.1 *Test scenarios for P2P architecture validation.*

Peer-to-peer technology enables any network-aware device to provide services to another network-aware device. A device in a P2P network can provide access to any type of resource that it has at its disposal, whether documents, storage capacity, computing power, or even its own human operator.

As it is explained on [1] the main advantage of P2P networks is that they distribute the responsibility of providing services among all peers on the network; this eliminates service outages due to a single point of failure and provides a more scalable solution for offering services. In addition, P2P networks exploit available bandwidth across the entire network by using a variety of communication channels and by filling bandwidth to the "edge" of the Internet. On the other side, P2P suffers from some disadvantages due to the redundant nature of a P2P network's structure.

### 4.1.1 JXTA peer2peer network.

One of the main tasks faced at the beginning of Phase II of the project, [2] was to look for a standard protocol that properly satisfies all the requirements from the partners and the SME's. The next task was to look for an open standard protocol that grants decentralization, flexible enough to follow the constantly changing requirements of Digital Ecosystems and with good support of security. After an analysis of the available open protocols and considering already deployed services, the selected network was JXTA.

JXTA was one of the evaluated protocols. JXTA peers create a virtual network where any peer can interact with other peers and resources directly, even when some of the peers and resources are behind firewalls and network address translations (NAT's) or on different network transports.

*Elements of JXTA Networks*

JXTA network infrastructure is composed of different peers:

**Simple Peers**: A simple peer is designed to serve a single end user, allowing that user to provide services from his device and consume services provided by other peers on the network.

**Rendezvous Peers**: They provide peers with a network location to use to discover other peers and their resources.

**Relay Peers**: Provide a mechanism for peers to communicate with other peers separated from the network by firewall or Network Address Translation (NAT) equipment.

*Peer groups*

A peer group is defined as a set of peers formed to serve a common interest or goal dictated by the peers involved. Peer groups can provide services to their member

peers that aren't accessible by other peers in the P2P network Peer groups divide the P2P network into groups of peers with common goals based on the following: *applications* they want to collaborate on as a group, the *security* requirements of the peers involved and the need for *status information* on members of the group

<u>*Layers*</u>
JXTA architecture [3] includes the Layers shown in the following figure.
*The JXTA platform*: Encapsulates minimal and essential primitives that are common to P2P networking like discovery, transport (including firewall handling), creation of peers and peer groups and associated security primitives.
*The Service JXTA Layer*: Includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in the P2P environment like: searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, and authentication with PKI (Public Key Infrastructure).

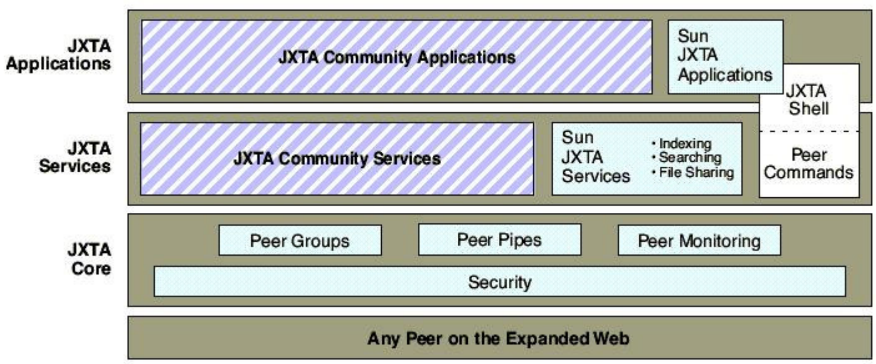



**Figure 2.- JXTA architecture.**

*The JXTA application layer:* Provides the implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P Email systems, or distributed auction systems.

### 4.1.1.1 <u>JXTA protocols.</u>

JXTA provides the protocols for basic functions of peer-to-peer networking, such as creating, finding, joining, leaving and monitoring groups, talking to other groups and peers, and sharing content and services. The functions are performed by exchanging XML advertisements and messages between peers.

#### *4.1.1.1.1 Peer Discovery Protocol (PDP)*

Peers use this protocol to discover all published JXTA resources. Since advertisements represent published resources, PDP essentially helps a peer discover an advertisement on other peers. As the lowest-level discovery protocol, PDP provides a basic mechanism for discovery. Applications might choose to use higher-level discovery mechanisms.

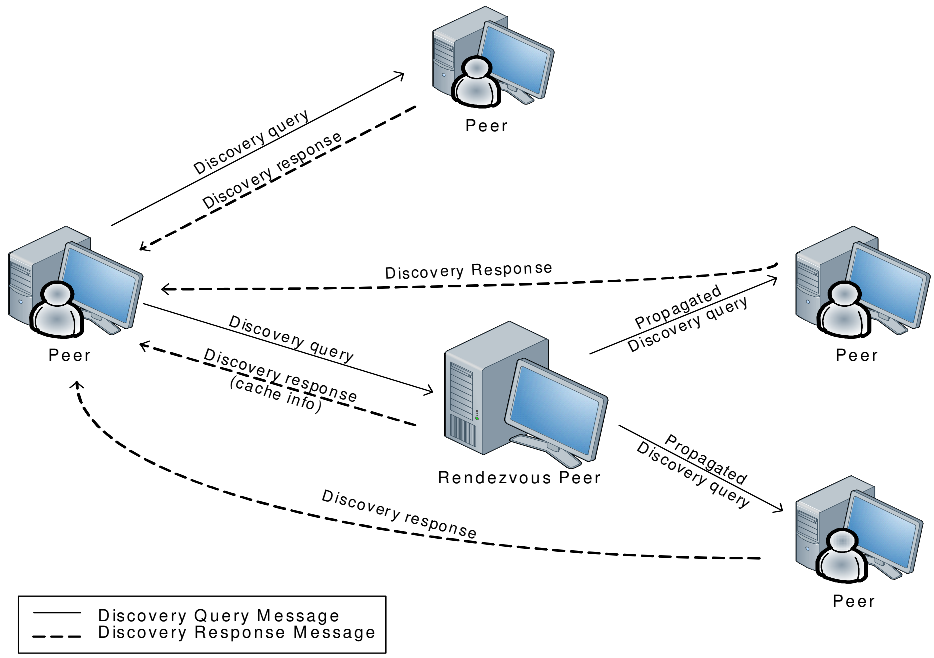



**Figure 3.- Peer Discovering Protocol Scenario.**

A peer can discover an advertisement in three ways, no discovery, direct discovery or indirect discovery which requires the use of a rendezvous peer.

All the protocols defined by the JXTA Protocols Specification are implemented as services called core services. The one related with this protocol is de Discover Service. By default only peers that are members of the same peer group are capable of communicating with each other via their services. The Discovery service provides a mechanism for the following: Retrieving remote advertisements, retrieving local advertisements, publishing advertisements locally publishing advertisements remotely or flushing local advertisements.

#### *4.1.1.1.2 Peer Resolver Protocol (PRP)*

Often in the network, peers send queries to other peers to locate some service or content. The Peer Resolver Protocol intends to standardize these queries' formats. With this protocol, peers can send generic queries and receive responses. It is responsible for wrapping a query string in a more generic message format and sending it to a specific handler on a remote peer.

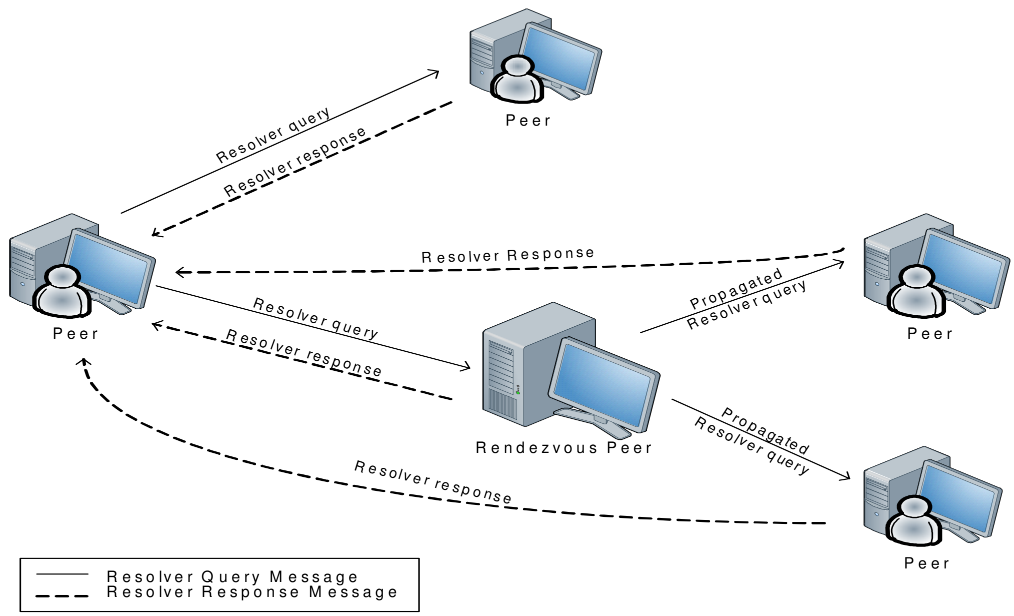



**Figure 4.- Peer Resolver Protocol Scenario.**

### *4.1.1.1.3 Rendezvous Protocol (RVP)*

The Redezvous protocol describes how messages are propagated peers in the member group. Before a peer can use a rendezvous peer to propagate messages, it must connect to the rendezvous peer and obtain a lease. A lease specifies the amount of time that the peer requesting a connection to the rendezvous peer is allowed to use the rendezvous peer before it must renew the connection lease.

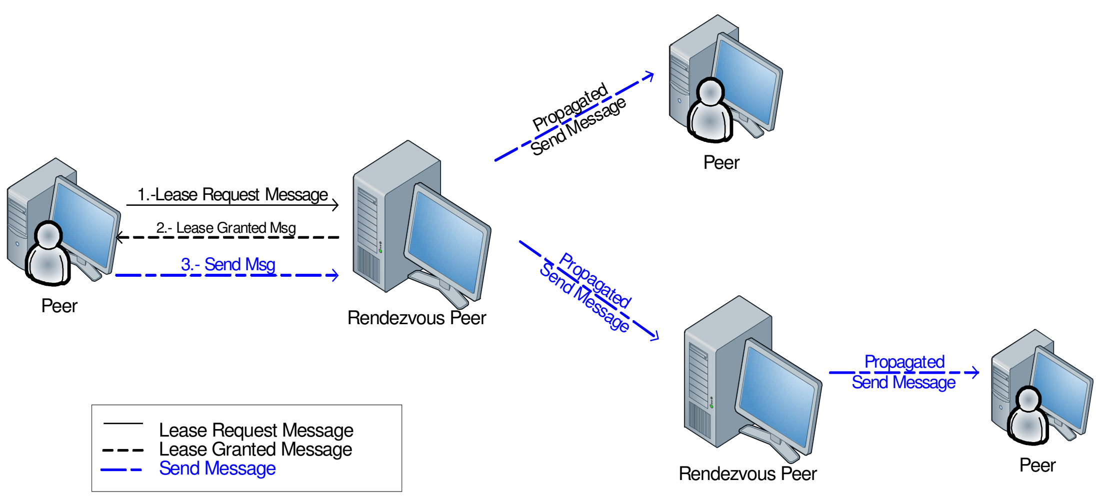



**Figure 5.- Rendezvous Protocol Scenario**

### *4.1.1.1.4 Peer Information Protocol (PIP)*

The Peer Information Protocol (PIP) is an optional JXTA protocol that allows a peer to monitor a remote peer and obtain information on the remote peer's current status. PIP can be used to "ping" a peer in the JXTA environment. A peer receiving a ping message has several options: It can give a simple acknowledgement, consisting only of its uptime. It can send a full response, which includes its advertisement. Or it can ignore the ping. Thus, there can be peers capable of receiving messages but not sending responses.

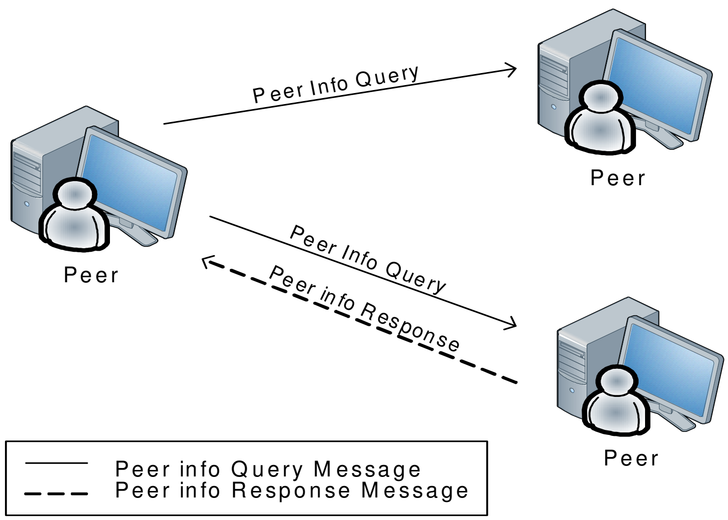



**Figure 6.- Peer Information Protocol Scenario**

### *4.1.1.1.5 Pipe Binding Protocol (PBP)*

In the JXTA environment, peers use pipes to access services. A peer can bind to a pipe's end at runtime and access services. The peer can create a new pipe, bind to an existing pipe, and unbind from a pipe. For those cases, the peer uses the Pipe Binding Protocol. Pipes are an abstraction in JXTA that describe a connection between a sending endpoint and one or more receiving endpoints.

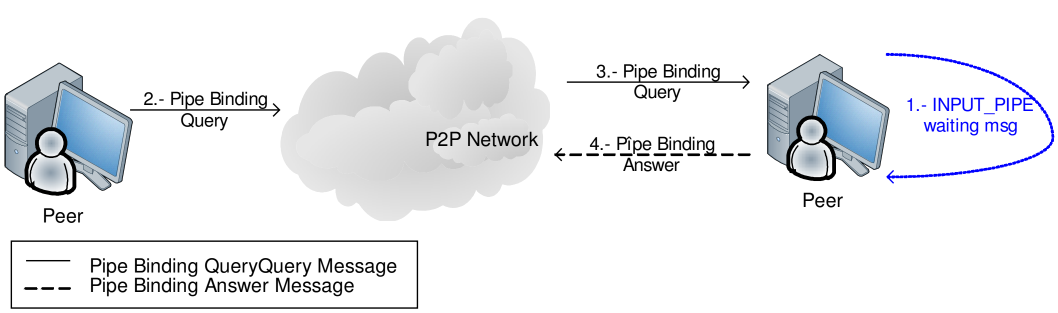



**Figure 7.- Pipe Binding Protocol Scenario**

### *4.1.1.1.6 Endpoint Routing Protocol (ERP)*

This architecture needs a mechanism to send messages between peers that aren't directly connected. If two peers cannot communicate directly using a common endpoint protocol implementation, the ERP provides each peer with a way to discover how it can send messages to the other peer via an intermediary, using only available endpoint protocol implementations. The Endpoint Routing Protocol, also called the Peer Endpoint Protocol, provides a mechanism for a message to be sent to a remote peer using discovered route information.

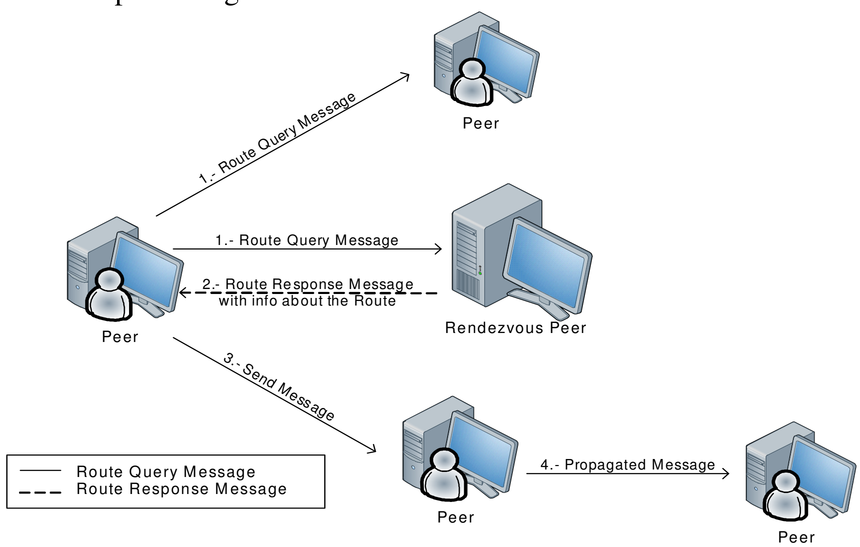



**Figure 8.- Endpoint Routing Protocol Scenario inside a group**

The previous scenario represents the ERP inside a peergroup, without the use of combined Firewall and NAT scenarios. The following one includes the use of route peers in order to send messages to remote peers from different networks.

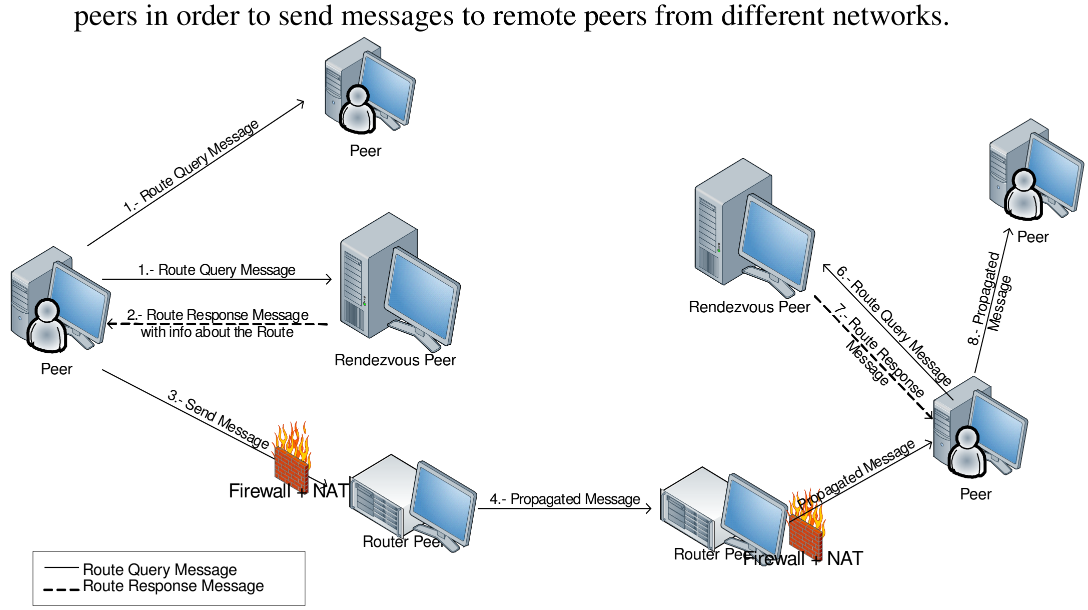



**Figure 9.- Endpoint Routing Protocol Scenario between remote networks**

### *4.1.1.1.7 Peer Membership Protocol (PMP)*

The peer membership protocol is a mechanism for joining peer groups. It imposes and verifies specific requirements for a peer to join a group. In other words, the protocol makes sure you give the right answers to questions that identify the peer as a valid group candidate before it's allowed to join a group.

Peers use the Peer Membership Protocol for joining and leaving peer groups. This protocol recognizes four discrete steps used by peers and thus defines JXTA messages for each of these actions: apply, join, renew and cancel.

The membership service is used to manage peergroup membership, and issue membership credentials. New peers need to be authenticated before they can join a peergroup. The membership service provides a pluggable authentication framework to support different authentication mechanisms (JAAS, LDAP).

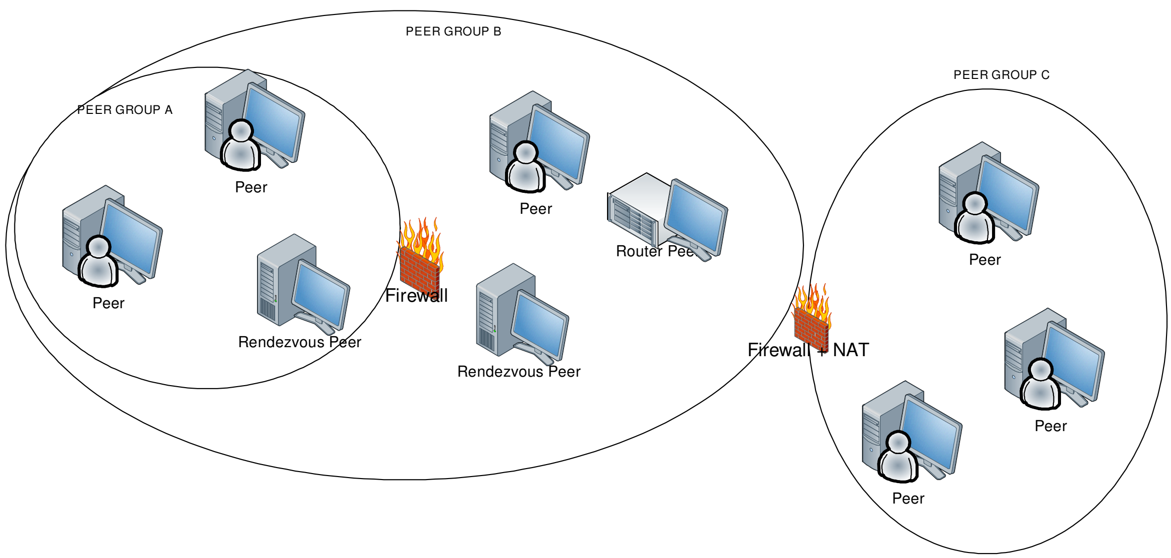



**Figure 10.- Peergroup membership scenario**

## 4.1.1.2 <u>Actions to be tested in a P2P architecture.</u>

All protocols previously defined are needed to test several types of interactions between peers. The scenarios described will be use to control the following actions.

### *4.1.1.2.1 Finding peers on the network*

In order to test this action the Peer Discover Protocol will be used, asking all knowing peers about their advertisements. The discovery of advertisements could be done with different methods: "no discovery", "direct discovery" or indirect discovery as it's explained on 4.1.1.1.1.

#### *4.1.1.2.2 Finding what services a peer provides*

In this case the Peer Resolver Protocol and its service will be used. As it is explained on 4.1.1.1.2, peers send queries to other peers to locate some service or content.

#### *4.1.1.2.3 Obtaining status info from peers*

In order to test this action, peers will use the Information Protocol 4.1.1.1.4. This is an optional protocol that allows a peer to monitor a remote peer and obtain information on the remote peer's current status. PIP can be used to "ping" a peer in the environment.

#### *4.1.1.2.4 Invoking a service on a peer / Creating data connections to peers*

With the use of the Pipe Binding Protocol, a peer will access to known services of others peers. As is it explained and shown in the scenario from 4.1.1.1.5, a peer can bind to a pipe's end at runtime and access services. The peer can create a new pipe, bind to an existing pipe, and unbind from a pipe. Creating, joining and leaving peer groups.

#### *4.1.1.2.5 Creating, joining and leaving peer groups*

In this case the Peer Membership Protocol 4.1.1.1.7 will be used in order to manage peergroup membership, and issue membership credentials. New peers need to be authenticated before they can join a peergroup.

#### *4.1.1.2.6 Routing messages for other peers*

In this case both Endpoint Routing Protocol (ERP) 0 and Rendezvous Protocol (RVP) 4.1.1.1.3 will be used in order to route messages to other peers. Whereas the RVP describes how messages are propagated peers in the member group, the ERP provides a mechanism for a message to be sent to a remote peer using discovered route information.

## 4.2 *Global Scenarios for a P2P Architecture Validation.*

This section includes a couple of global scenarios as a reference to do the peer2peer platform validation. The first one, include two internal networks connected through a router. In each of the networks some rendezvous peers and simple peers will be included, as many as necessary for each of the defined tests.

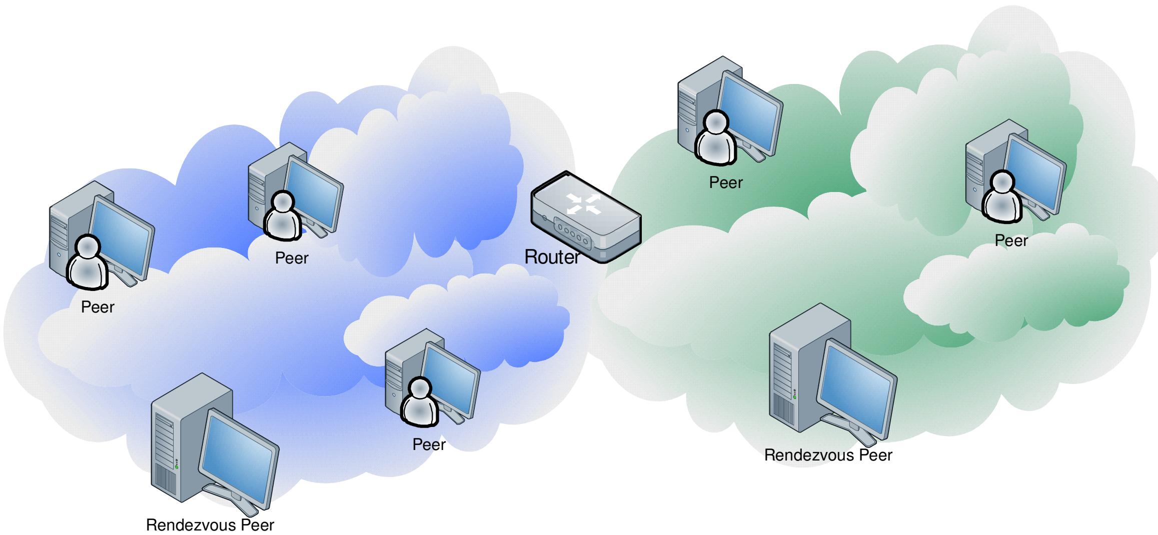



**Figure 11.- Global scenario in local networks**

The previous scenario will help to test several actions like the following:

-Finding peers on the network
-Finding what services a peer provides
-Obtaining status info from peers
-Invoking a service on a peer / Creating data connections to peers
-Routing messages for other peers internally

The following scenario will include external peers emulating the internet and the normal connectivity trough the use of firewall and Nat. In each of the networks some rendezvous, peers, router peers and simple peers will be included, as many as necessary for each of the defined tests The following actions will be able to be tested under it:

-Routing messages for other external peers
-Creating, joining and leaving peer groups

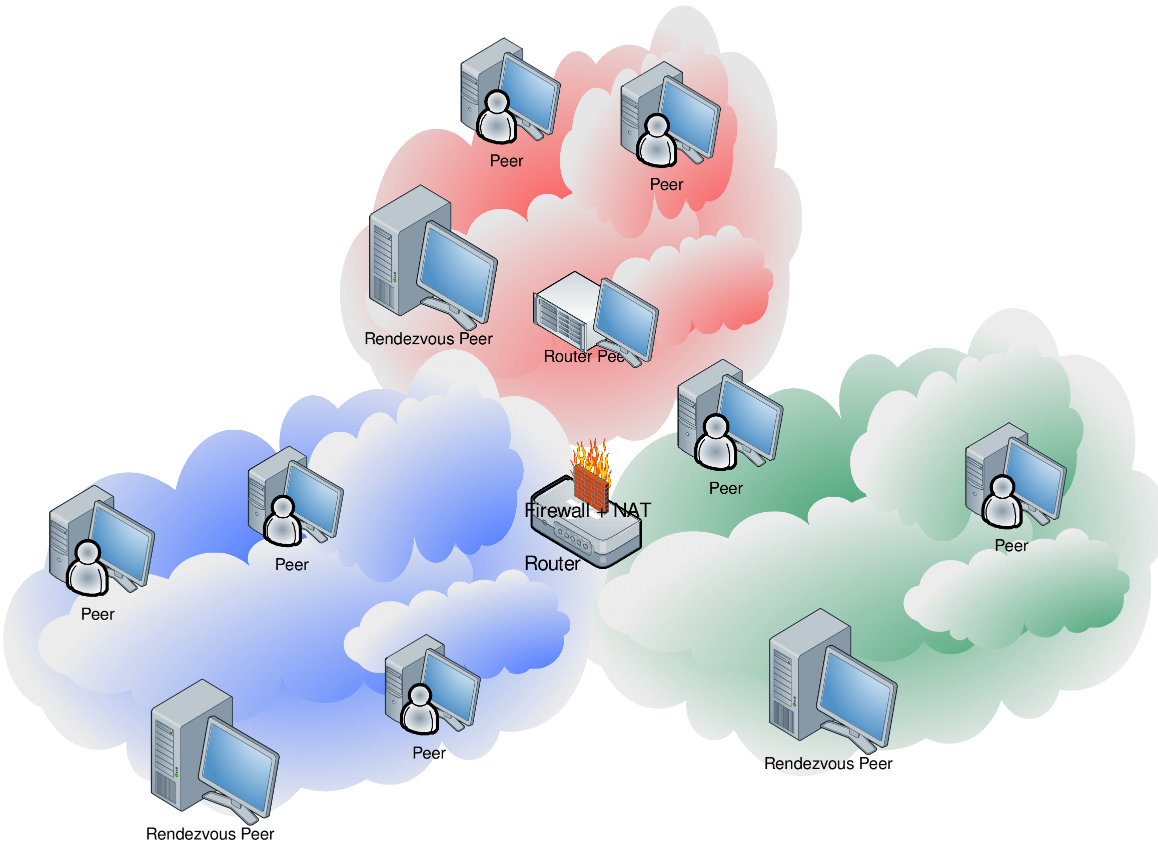



**Figure 12.- Global scenario in local and external networks**

## 4.3 *Requirements and Test scenarios for P2P system validation.*

### 4.3.1 General requirements

This section describes general requirements that a P2P system architecture must and/or should guarantee.

1. *Robustness*: Any system must be capable of withstanding errors which should not affect system stability.
2. *Reliability*: This implies the continued or regains balance. It's the ability to recover from an error in the system and return to the previous state before the error occurred.
3. *Security and confidentiality*: The system must be protected from unauthorized access to the system and must be able to not allow personal data to be read or modified by unauthorized persons.
4. *Scalability and Extensibility*: The system must allow support for a variable number of users.
5. *Integration*: The system must have the ability to communicate with other systems that are in different architectures.
6. *Management and Provisioning*: The system must ensure its management and monitoring of implemented services.

The following requirements should also be guarantee in this kind of distributed system.

1. *Decentralization*. These networks are by definition decentralized p2p and all nodes are equal. There are no nodes with special functions, and therefore no node is essential for the operation of the network.
2. *Based on open standards*.
3. *Interoperability*: This requirement is defined as the ability of a system or product has to work with other systems or products without special effort.
4. *Portability*: It is possible that the application may be available on all machines regardless of the system architecture.
5. *Availability*: To be freely available or that it is ready for use or used. The system should be available as long as possible.
6. *Persistent*: The system should ensure the ability to store information of the system to return to the previous or retrieve information.
7. *On time*: The response of the system should be given within an appropriate timeframe.
8. *Reliable or deterministic*: The system should give the same result while making a deal with the same operators in the same context of operations.
9. *Transactional*: The system should be able to return to its state before the transaction started.

10. *Modifiability:* Modifiability is about the ease with which a change can be made to an application architecture. A particular concern of network-based systems is dynamic modifiability where the modification is made to a deployed application without stopping and restarting the entire system. It includes requirements like *evolvability, extensibility, customizability*, *configurability* and *reusability*.
11. *Support for extended web services protocols* like *Decentralization, Security, Flexibility, Ubiquity* or *Extensibility***.**

### 4.3.2 Requirement extract of previous deliverables.

This chapter includes all requirements previously defined in other OPAALS deliverables. They all have been defined as important components to be guarantee in the final p2p system. In the following sections some of them will be included on the test plan in order to validate the system.

#### 4.3.2.1 OPAALS release 1 Transaction Model

This concept has been included on several deliverables.

◘ As it is explained on Deliverable D3.2 point 2.1 [5] one of the requirements is ***the transaction support*** which is based on the following properties:

- Atomicity: either all tasks in a transaction are performed, or none of them are.
- Consistency: data is in a consistent state when the transaction begins, and when it ends.
- Isolation: all operations in a transaction are isolated from operations outside the transaction.
- Durability: upon successful completion, the result of the transaction will persist.

However, in advanced distributed applications these properties can present unacceptable limitations and reduce performance.

The specification of a *transaction* may involve a number of required services, from different providers, and allow it to be completed over a period of minutes, hours, or even days – hence, the term long-lived or long-running transaction. This makes the adoption of the Service-Oriented Computing (SOC) more relevant than ever. The goal of SOC is to enable applications from different providers to be offered as services that can be used, composed and coordinated in a loosely-coupled manner. This means that a service must be designed in a way that it can be invoked by various service clients and is logically decoupled from any service caller. The actual architectural approach of SOC is called SOA.

◘ On Deliverable D3.1 point 3.3 [6] is explained the **Distributed Transaction Model for OPAALS**.

In this section it is explained the proposed model for distributed long-running multi-service transactions in OPAALS. The autopoietic P2P network is to support a healthy and diverse socio-economic ecosystem that is the primary “business goal” of the OPAALS project. From that comes a specific focus on supporting and enabling a service-oriented business environment which facilitates business

transactions between participating SMEs in a loosely coupled manner without relying on a centralized provider.

Below are the pre-conditions and motivations described for the model which were designed for transactions between open communities of SMEs in digital ecosystems for business.

**-** *Long-term transactions***:** A high range of B2B transactions have a long execution time period, strictly adhering to ACID properties for such transactions can be highly problematic and can reduce concurrency dramatically. The application of a traditional lock system (as the concurrency control mechanism for ensuring Isolation, or capturing some version of serializability reduces concurrency and the general performance of the whole system – many transactions have to wait for a long-term transaction to commit and release its resources or results.

**-** *Partial results:* Releasing results from one transaction to another before either transaction commits is another challenge in the distributed transactional environment. According to conventional transaction models, releasing results before transaction commit is not legal, as it can misdirect the system to an inconsistent state when the transaction is aborted before final commit and all released results are not valid anymore. It may be argued that this situation does not happen regularly, but it is a crucial requirement which has to be automated.

**-** *Recoverability and failures***:** Recovering the system in the event of failure or abortion of a transaction needs to be addressed in a way that takes into account the loosely-coupled manner of connections. This makes a recoverability mechanism in this context even more challenging. As we can not interfere with the local state of the underlying services, the recovery has to be done at the deployment level and service realization (which includes the state of a service) has to be hidden during recovery.

**-** *Diversity and alternative scenarios***:** The provision for diversity has been referenced as the system has integrated SMEs with rather diverse environment for business transactions. When considered at the transaction model and/or business processes, it can provide a unique opportunity in not only covering a wider range of business processes but also in designing a corresponding recovery system.

**-** *Recovery - omitted results***:** It has been discussed how to preserve as much progress–to-date as possible. Raising to this challenge within a highly dynamic environment such as DBE can have significant direct benefits for SMEs in terms of saving time and resources.

**-** *Levels of centralization***:** In different transactional models and workflow managers, there exist varying levels of centralization by using centralized coordinators. Inherently this causes limitations on composition, compromises the ability for automating different types of compositions reduces platform autonomy and the range of partial results Therefore, with respect to additional architectural

complications, the system should apply to a fully distributed structure, which enables full autonomy of the local platforms.

**-** *Platforms failures and contingencies plan***:** Platform failures are one of the natural events in web services that can easily cause a transaction to fail, but according to the user specification request, range of accessible alternative web services or even business model, which triggered the transaction, most of these incidents should not cause the transaction to fail.

◘ On Deliverable D3.1 point 5 [6] is explained the **A Peer-to-Peer Network for Digital Ecosystems**

In this section it is explained the transactional model the *autopoietic* P2P network (as a infrastructure of digital business ecosystem network) is intended to support and the following requirements.

**-** *Consistency***:** This system is designed for the needs of business, and not just the sharing of information within a community. Within a single business transaction, there may be a set of sub-transactions involving different organizations. Hence, the P2P network must support a *highly transactional* environment which can cause huge amount of traffic and support *long-running transactions* which need replication of critical system logs.

- *Latency*: It is a critical impact factor on the performance of the infrastructure in supporting long-running business transactions.

- *Local autonomy*: Although the autopoietic P2P network as a whole must belong to the community to enable their services to be freely published and discovered, the autonomy of individual nodes within the network must be respected. Hence, the system should guarantee that: There must be *no limitation to the ownership* or decision making of a single node's services.

- *Reliability*: The system should handle with address/manage failure risks for any infrastructural services.

- *Security*: The network primary structure must fully support development of a secure transaction model

-*Availability*: Consistent replication

- *Boundaries*: In principle there should be no boundaries to the growth of the network and must be a scale-free network. The system requires a dynamic structure to handle bandwidth and load balancing issues.

- *Levels of centralization*: As a technical requirement, a fully distributed infrastructure is required, which means it should not be centralized (including clusters with strong dependencies) and cannot be limited to decentralization.

- *Bootstrapping and Scalability*: The system should ensure *accessibility* which is the ability for new members to register with and join into the network and scalability, ensuring critical properties are preserved as the network grows

- *Maximum Distance*: Specifying the diameter according to the lookup algorithm.

- *Durability* High reliability of the whole network and a reliability model for the network to understand what guarantees can be given.

- *De-fragmentation algorithm:* One of the serious dangers for a distributed network is that of fragmentation of the network into separate sub-networks. In this situation, peer-to-peer connections between different nodes, running different transactions and queries, may fail. That's why the system should be self-healing and have the ability for recovering after failure happened. It also needs a good theoretical understanding of the boundaries for recovery.

- *Replication Model:* This replication model should be consistent and should support delegation to enable roles or responsibilities to be rapidly delegated should a node be partially or wholly unable to fulfill them for some period of time.

◘ On Deliverable D3.1 point 5.5 [6] are included **Other important issues.**

In this section, additional dimensions of the autopoietic P2P network for OPAALS were discussed including new requirements included behind. More specifically, it's explained topics of distributed identity and trust since these aspects are necessary for fostering a collaborative environment that facilitates the sustainability of the digital ecosystem.

- *Distributed trust*: In P2P systems, peers often must interact with unknown or unfamiliar peers, who may even reside in different security domains, and a major challenge is how to establish trust between different peers. By and large, the purpose of trust in this context is to protect against those who offer services rather than from those who want to access them.

- *Distributed identity*: In terms of avoiding the bottleneck of dependency to a central authority is one of the most important issues in today's large scale networks, especially when the uniqueness of the identity is considered as the key point of the network. Further more according to the nature of this system, not only each node needs a unique address but also each transaction has to have a unique identity too. During the process of delegation these two unique identity play crucial role for consistency of the model.

◘ On deliverable D3.6 point 4.3 [7] are included requirements for **A Peer-to-Peer Network Design for DEs.**

The main objective in the first instance of the network design is to support long-lived transactions. The Basic requirements for the digital ecosystem network are that it must be distributed (a premise for lowering the barrier of adoption), resilient to failure (to perform transactions in a distributed manner), and have a dynamic topology that continuously evolves to reflect the changing needs of the interacting communities it supports. The *peer-to-peer* (P2P) network design exhibits such features and enhances the ability of the DE core architecture to address issues of power and control as well as avert monopoly phenomena and ultimately protect the democratic nature of DEs.

It is also described the **P2P network of connected VTPNs: further challenges** where outlined key aspects that have gone into our P2P, designed to provide a fully connected network for the DE core architecture. The basic idea behind the design is to use the local interactions that take place between participants in long-running transaction as the main building block for the overall P2P network. These local interactions between peers in a similar domain come with the characteristics of interactions within a cluster and since they are part of a transaction they also come with useful information that can be exploited in providing a DE architecture that is purely distributed, dynamic, self-organizing and highly resilient to failure

◘ On deliverable D3.6 point 4.3 [7] are included requirements for **Identity, Accounting, and Trust in the OPAALS DE**

- *Decentralized:* One of the fundamental requirements for digital ecosystems is that there is no single point of failure. If any one node (or any sets of nodes) becomes unavailable, the system must be capable of recovering completely without any external intervention. A suitable accountability solution requires therefore that there is no dependence on a single centralized accountability manager or coordinator and that all functionality is distributed across the system.

- *Service Composition:* A crucial aspect of digital ecosystems is collaboration between participants. In this sense it is important that services and resources can be combined to create new services. This service composition needs to be dynamic and cannot be known in advance. An accountability solution requires that such a dynamic composition of services can be seamlessly and efficiently accounted for.

- *Scalability:* The number of peers in a digital ecosystem is arbitrary, ranging from a handful to several thousand. A suitable solution needs to work for large sets of peers as well as large service compositions.

- *Security:* The implications for security when providing accountability in dynamic composing in digital ecosystems are wide ranging. Integrity of accounted data, availability of accounted data, confidentiality of accounted data, privacy of

transactions, all need to be considered when designing a suitable accountability model.

- *Contracts:* Exchange of contracts or service level agreements prior to service consumption is a vital aspect of a commercial application of digital ecosystem deployment. It is desirable that aspects of these agreements can be assessed at runtime to ensure that parties are operating within the bounds of the agreement.

### 4.3.3 Requirements defined for the *Re-engineered Servent*

This section summarizes the requirements defined for the Re-engineered Servent that must, should or won't be able to validate in the system. For a more concrete description please take a look at the D5.3 document [2].

First table introduce the requirements the infrastructure **must** perform.

| Rid | R1 Requirements: the infrastructure must... |
|---|---|
| **R1.1** | Be service oriented. |
| **R1.2** | Manage service deployment. |
| **R1.3** | Manage service searching. |
| **R1.4** | Manage service consuming. |
| **R1.5** | Allow service composition |
| **R1.6** | Manage security in the transport layer. |
| **R1.7** | Manage identity. |
| **R1.8** | Make bootstrapping connection easier. |
| **R1.9** | Manage transactions in service composition. |
| **R1.10** | Give support for scalability and performance. |

**Table 13.- Priority Requirements.**

All these requirements are a must in this kind of P2P networks, some of them have been described on previous deliverables - *Deliverable D3.6 point 2.1.5.*

**R1.1** *The infrastructure must be service-oriented*. Here, a *service* is understood as a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. This is in fact the definition of a service as given by the Organization for the Advancement of Structured Information Standards (OASIS). Usage of a service must only require access to a prescribed interface and not to the data it uses and its implementation details.

**R1.2**. *The infrastructure must manage service deployment*, i.e. the service life-cycle from starting to stopping. It must also manage service searching for the services employed into it. It must also manage the remote execution of services deployed into it, given a known or standard interface.

**R1.3 – R1.4** *The system must manage service searching and service consuming*, providing the information related to available services and it's consuming.

**R1.5**. *The infrastructure must allow for service composition* between the services deployed into it. When read in conjunction with requirement R1.1, the infrastructure must support interaction-based model of service composition, i.e. through calls to prescribed interfaces. The survey also showed that dynamic service composition is desirable but can only be applied in restricted areas and dynamic business transactions require a legal or regulatory framework.

**R1.6**. *The system must manage security* in the transport layer, which means the use of services is done through a secure infrastructure giving support for extended web services protocols

**R1.7** *The infrastructure must manage identity* Any user entity on the infrastructure must be given an identity so that all peers can be identified when engaging in interactions (e.g. business transactions, knowledge services) so that it can be accountable for its actions.

**R1.8** *The infrastructure must make bootstrapping connection easier* in order to add new peers in the network.

**R1.9**. *The infrastructure must manage transactions in service composition*. In the composition of services a transaction manager is needed to coordinate the underlying services deployment of distributed transactions.
The transaction manager must also coordinate the rollback mechanisms that can be executed in case of failure. The transaction manager must manage service compositions and rollback mechanisms without requiring access to the actual state of service execution.

**R1.10**. The infrastructure must give support for *scalability and performance*. Even though there is no agreement on figures to determine performance or scalability, it is under common sense that infrastructure must be scalable and offer good performance. Further interviews with SMEs showed that performance is often represented by response time when searching or executing. Scalability is represented by the challenger of adding new users or services without a high performance reduction.

The following table summarizes the requirements the platform **should** implement although they were defined to have less priority than the listed in the previous table.

| Rid | R2 Requirements: the infrastructure should... |
|---|---|
| **R2.1** | Be based on web services standards. |
| **R2.2** | Give support for semantic management. |
| **R2.3** | Run on different software platforms. |

| | |
|---|---|
| **R2.4** | Run on different hardware platforms. |
| **R2.5** | Be integrated to other infrastructures. |
| **R2.6** | Be based on JBI and OSGi standards. |
| **R2.7** | Support for asynchronous communication. |
| **R2.8** | Support for extended web-services protocols. |
| **R2.9** | Manage distributed identity. |
| **R2.10** | Implement some self-* properties. |
| **R2.11** | Use P2P network architecture. |
| **R2.12** | Give support for semantic management. |

**Table 14.- Requirements the infrastructure should implement.**

Although all these requirements should be implemented in this kind of P2P networks, only some of them have been selected as desirable for this system.

**R2.1**. *The infrastructure should be based on standards* and give support for semantic management.

**R2.3 – R2.4**. *The infrastructure should run on different software and hardware platforms*.

**R2.7**. *The infrastructure should have support for asynchronous communication*.

**R2.10**. *The infrastructure should implement some self-*properties*. In view of its use for connecting SMEs performing business transactions, the aim should be for a reliable platform that can handle failures, which means a platform that is capable of self-management, self-organizing and other self-properties such as:

- self-healing; automatic discovery and correction of faults
- self-optimization; automatic monitoring and control of resources to ensure optimal functioning with respect to defined requirements
- self-protection; proactive identification and protection from arbitrary attacks.

**R2.11**. *The infrastructure should use P2P network architecture*. This is necessary so that every peer can operate under the same conditions and no SME can exploit the basic infrastructure used by all to gain a dominant position in the market. In the remaining chapters of this report we describe the key aspects of the core architecture, in terms of the P2P network, transaction support and trust and identity.

Finally, the last table include the list of requirements that can not be implemented by the infrastructure but must be considered.

| Rid | R3 Requirements that can't be implemented into the infrastructure. |
|---|---|
| **R3.1** | **Legal Regulatory Framework.** |
| **R3.2** | **Quick Bootstrapping.** |
| **R3.3** | **Have a support community.** |
| **R3.4** | **Have an Integrated Development Environment.** |

**Table 15.- Requirements the infrastructure cannot implement.**

### 4.3.4 Selected requirements for P2P Platform Validation Test Plan.

This section include all important requirements that have been introduced in this chapter 4.- *Requirements and Tests for P2P Platform Validation* – that are going to be part of the test roadmap.

To begin with, we include most of the requirements defined for the *Re-engineered Servent* as a "must" and some of the ones classified as "should be implemented" are included as final P2P Platform Validation requirements as they were adopted in previous implementations of the platform

| R_ID | ***Selected requirements.*** |
|---|---|
| **R_1** | The infrastructure must be *service-oriented.* |
| **R_2** | The infrastructure must *manage service deployment* |
| **R_3** | The system must manage service searching and service consuming – *management and provisioning* |
| **R_4** | The infrastructure must allow for service composition |
| **R_5** | The system must manage security –*security* |
| **R_6** | The infrastructure must manage identity – *confidentiality* |
| **R_7** | The infrastructure must make bootstrapping connection easier. |
| **R_8** | The infrastructure must give support for *scalability and performance*. |
| **R_9** | The infrastructure should be based on *standards* |
| **R_10** | The infrastructure should run on different software and hardware platforms – *portability* |
| **R_11** | The infrastructure should have support for asynchronous communication |
| **R_12** | The infrastructure should implement some self-healing, self-optimization and self-protection properties |
| **R_13** | The infrastructure should use P2P network architecture – *decentralization* |

After reviewing general and other requirements explained in previous OPAALS deliverables, the following requirements have been selected as important issues to be tested.

| R_ID | ***Selected requirements.*** |
|---|---|
| **R_14** | The infrastructure must ensure *robustness*. |
| **R_15** | The infrastructure must ensure *reliability*. |
| **R_16** | The system must provide integration with systems that are on different architectures offering *interoperability*. |
| **R_17** | The infrastructure should be *transactional*. |

## 4.4 *Test definition.*

### 4.4.1 Test definition process.

This section includes the definition process of the tests that are going to be implemented in order to test the selected requirements for a p2p platform validation. It is described for example, the way to name test, way to name the errors, way to report test/bugs evolution, evaluation criteria and state, definition, objectives, type of test and so on.

**Naming of the Test.**

As the project include several Releases, it will be used the name (R0, R1 …) of the first release it is going to be run together with a "R" referring to the requirement, followed by an incremental number which will be restarted with each release. Some examples are:

- R0.R01: the first performed test of the first release.
- R4.R01: the first test of the fourth release.
- R5.R28: the test 28 of the fifth release.

**Description**

This section will include the relevance of passing the evaluation test. It's important to know the description of the requirement that is going to be evaluated for each release, as in some releases, maybe the test won't apply.

**Objectives**

Describes the main issue of doing this kind of test.

**Type of test**

Include a description of the way of doing the tests, in terms of manual or automatic testing.

**Architecture**

Select the architecture or scenario needed for evaluating the software and the selected requirement. Most of the scenarios are described on 4.2.

**Test Suite case**

The test suite is a collection of test cases that are intended to be used to test a software program to show whether or not it passes the selected requirement. It is implemented as a class, which is the main class of the testing application and it contains several test cases, which are implemented as a set of actions that are implemented as annotated methods.

**Naming of the Reported Bugs.**

While executing the described tests, some bugs will appear. In order to identify them with the test definition, it will be used the same naming schema as with the name of the test.

The bug Identitfier will begin with the release number (R0, R1, …) followed by the use of "D" if the errors refer to the documentation or "S" if the error is a software bug, Finally an incremental number will be included which will be restarted with each release. Some examples are:

- R0.D. 01: the first documentation reported bug for the first release.
- R0.S.02: the second reported bug which refers to a software bug for the first release.
- R5.S. 28: the 28th reported software bug of the fifth software release.

**Evaluation of the Platform against its Requirements.**

In the evaluation process it will be included the state of the evaluation, identifying the following ones:

- *Not Valuable*: The requirement is outside the scope of the development team. It will depend of the release and the requirement definition.
- *Not passed*: The test has been run but some bugs have appeared.
- *Passed*: the requirement is completely implemented and without bugs.

**Bugs Reporting and Tracking.**

While making tests evaluation, normally some bugs are found, that could help in the following releases deployment. This section will include the *test closure actions* and the *actions to be performed if test fails in order to solve the problem.* Also the impact on the final integration and validation process will be described.

### 4.4.2 Test definition for each of the selected requirements.

This section includes the definition of the test process related to selected requirements in order to validate the peer-to-peer platform (4.3.4). Each of them will be described with its *name test case*, the *description*, *objectives*, *type of test* and *test suite case* fields.

#### 4.4.2.1 The infrastructure must be service-oriented.

Name test case
RX.R01

Description
Involve a number of required services, from different providers, and allow it to be completed over a period of minutes, hours, or even days – hence, the term long-lived or long-running transaction. This makes the adoption of the Service-Oriented Computing (SOC) more relevant than ever.

The goal of SOC is to enable applications from different providers to be offered as services that can be used, composed and coordinated in a loosely-coupled manner. This means that a service must be designed in a way that it can be invoked by various service clients and is logically decoupled from any service caller. The actual architectural approach of SOC is called SOA.

Objectives
Check that a service can be built from different suppliers. These services should be decoupled built, which means that the tester can switch supplier, being transparent to the final user.

Type of test
A set of automatic tests will be created in order to validate this characteristic.

Test Suite case
It is defined with the following tests:

- *Test_1*: *Service composer, coordinated*: One service is composed of nine different services, which are deployed in 3 different nodes.
- Test_2: *Long-running transaction*: One service is composed ofthree services. One of them must wait two minutes until the information is submitted.
- Test_3: *Loosely-coupled*: One service is composed of three services. The service is the consumer. Then, the service provider is changed to another node but with the same the same service signature.

#### 4.4.2.2 The infrastructure must manage service deployment.

Name test case

RX.R02

Description

This test will validate de management of the service life-cycle, from starting to stopping giving also a known or standard interface.

Objectives

The management is an important part of the product whose mission is the success of a goal. An management infrastructure provides some functionality as: search, deploy, undeploy, change state of service, execute a service and ask service state. In order to test this management an API will be provided to execute automatically these operations.

Type of test

Check each test defined in the manual of FlyPeer related to service management deployment.

Test Suite case

- Test_1: This test will evaluate whether is possible or not to manage the deployment and undeployment of services automatically without going through the stop and restart steps.
- Test_2: This test will check if it's possible to manage changed state to start, pause, continue, restart and stop in a service.
- Test_3: This test will evaluate whether is possible to see the state of service for example, running, stopping or on pause.
- Test_4: This test will check it it's possible to create a program that makes all this operations.

#### 4.4.2.3 The system must manage service searching and service consuming – management and provisioning.

Name test case

RX.R03

Description

This test will evaluate the management of searching for the services employed into it. It must also verify the management of remote execution of services deployed into it, given a known or standard interface.

Objectives

The management is an important part of the product whose mission is the success of a goal. It provides some functionality as: search, deploy, undeploy change state of service, execute a service and ask service state. In order to test this management an API will be provided to execute automatically this operations as well.

<u>Type of test</u>
Check each test defined in the manual of FlyPeer related to manage service searching and providing.

<u>Test Suite case</u>

-Test_1: This test will check if it s possible to search the services.

-Test_2: This test will validate whether is possible to consume a service, previously found.
-Test_3: This test will check if it is possible to search a service with regular expressions or patters, for example it will check if it's possible to find a service, with will card (*) or keyword and use it.
-Test_4: This test will evaluate the execution of a remote service. - Test_5: This test will check it it's possible to create a program that makes all this operations.

#### 4.4.2.4 <u>The infrastructure must allow for service composition.</u>

<u>Name test case</u>
RX.R04

<u>Description</u>
A crucial aspect of this kind of systems is collaboration and interaction between participants. In this sense it is important that services and resources can be combined to create new services. This service composition needs to be dynamic and should not be known in advance.

<u>Objectives</u>
For the composition of services, the following steps must be followed.
Search the service, choose the one that best suit with the needs of the user, and execute the service remotely.

Each service has two types of class composition:

- *Static service composite*: In this case, the user must choose the service that best suits its needs. The user should know service description, a system that identifies each of the parameters and know the semantic meaning. After selecting the data type and order of the parameters he will create the adapter and finally execute the service.
- *Dynamic service composite*: In this case, all operations performed by the user through code in static mode, must be obtained through the system itself. The choice of services and its composition must be obtained through a set of rules.

<u>Type of test</u>
Check the type of tools given for the creation and composition of services.

Test Suite case

The following tests are defined in the test suite.

- Test 1: Does the system provide semantic meaning parameter?
- Test 2: Does the system provide conversion units?
- Test 3: Does the system provide adapters?
- Test 4: Check if a simple static service composite can be created.
- Test 5: Validate if a simple dynamic service composite can be created.
- Test 6: Check if a complex static service composite can be created.
- Test 7: Check if a complex dynamic service composite can be created.

#### 4.4.2.5 The system must manage security.

Name test case

RX.R05

Description

The implications for security when providing accountability in dynamic composing in digital ecosystems are wide ranging. While designing a suitable accountability model the following characteristics should be taking into account: integrity, availability, confidentiality and privacy of transactions.

Objectives

In order to check security in a system, the following issues will take into account.

- Protection of data to avoid unauthorized access.
- Systems won't allow read or modify data to unauthorized persons.
- Systems will provide access to authorized persons.
- The system should protect itself from accidental or malicious accesses.

The system security also include user accesses, secure data, secure communications and physical protection to avoid computer manipulation.

Type of test

This test will check what types of tools for the creation and composition of static and dynamic services are provided by the system.

Test Suite case

-Test_1: HTTP packets will be analyzed in order to check the level of security implemented by the system. It will be check whether private information like login, password, or private data could be shown.

#### 4.4.2.6 The infrastructure must manage identity – confidentiality.

Name test case
RX.R06

Description
To protect and preserve the confidentiality of information and data means to ensure that it is not available or disclosed to unauthorized entities. That means that the system must ensure that information is accessible only to those authorized to have access. In this context, entities include both individuals and processes.

Objectives
The system cannot let unauthorized persons read or modify data. As it has been included on the previous test, system security also includes user access, secure data, secure communications and physical protection to avoid computer manipulation.

Type of test
Check the identity and accounting access manager services and its confidentiality.

Test Suite case
- Test 1: Check the system verifies the identity and accounting access to services and test when an unauthorized entity attempts to access a secure service.

#### 4.4.2.7 The infrastructure must make bootstrapping connection easier.

Name test case
RX.R07

Description
In case of using p2p network technologies, the system must manage the connection to other new peers without manual administration. At the beginning, a stable community of peers can be constantly running so that they guarantee the first connection for bootstrapping. This should be extended with discovering algorithms in future.

Objectives
Once the installation and configuration of the application are finished, for each bootstrapping of the application check whether the management operations are repeated or not. The main issue will be to evaluate if new peers could easily launch and access the applications without the need to configure by themselves the service.

Type of test
This test will check the application document and will execute it using bootstrapping.

Test Suite case

The following test will be validated when the application has previously configured to access to the network and when it has not been configured.

- Test_1: At the bootstrapping, check that the application can connect to the network.

#### 4.4.2.8 The infrastructure must give support for scalability and performance.

Name test case
RX.R08

Description
As it's included on [2],it is under common sense that infrastructure must be scalable and offer good performance

- *Performance* is represented by response time when searching and executing a service. It's agreed that ten seconds is a top value.
- *Scalability* is represented by the challenge of adding new users or services without a high performance reduction (at least based on lineal or logarithmic reduction) and, more important, the absence of manual reconfiguration.

Objectives
The main issue of this test will be to evaluate the scalability and performance in terms of services and nodes.

Type of test
This test will be executed checking scalability and performance.

Test Suite case

- Test_1: Configure nodes to deploy a large number of services, so that scalability of services can be tested. As a result, the measurements performance and response of each of the requests will be evaluated.

- Test_2: In order to check scalability in terms of nodes and architecture a large number of nodes will be simulated in one of them. As a result , the measured performance and response of each of the requests will be considered.

#### 4.4.2.9 The infrastructure should be based on standards.

Name test case
RX.R09

Description
As it is introduced in [6] to attain interoperability in the context of pan-European services, guidance needs to focus on open standards. The following are the minimal characteristics that a specification and its attendant documents must have in order to be considered an open standard:

- The standard is adopted and will be maintained by a not-for-profit organisation, and its ongoing development occurs on the basis of an open decision-making procedure available to all interested parties (consensus or majority decision etc.).
- The standard has been published and the standard specification document is available either freely or at a nominal charge. It must be permissible to all to copy, distribute and use it for no fee or at a nominal fee.
- The intellectual property - i.e. patents possibly present - of (parts of) the standard is made irrevocably available on a royalty free basis.
- There are no constraints on the re-use of the standard.

Objectives
The system should be based on open solutions, which means on open standards. To test this feature, specified list of items to be verified by the software.

Type of test
The software will be checked to verify this requirement.

Test Suite case
- Test_1: Test that all interface are create and publish under open licence any organization.

#### 4.4.2.10 The infrastructure should run on different software and hardware platforms – portability.

Name test case
RX.R10

Description
This test will validate if it's possible to run de system on different software platforms like Windows, Solaris and Linux. Also it's important to check if it's possible to run on different hardware platforms, this involves a more difficult requirement with a higher added value, specially considering on mobile devices hw.

Objectives
The objective of this test is to verify that the language on which the application is built supports the vast majority of current operating systems.

Type of test
It will be verified manually.

Test Suite case
Test_1: Check this characteristic with the manual installation, and the software features vendor.

#### 4.4.2.11 The infrastructure should have support for asynchronous communication.

Name test case
RX.R11

Description
Most current web services applications are used in a connection-oriented way so that synchronous connections are required between the service requester and service provider. On the other side, there may be users that cannot be connected always, for this audience the support for asynchronous transactions may be an added value.

Objectives
The main issue of this test is to validate asynchronous communications, where the sender and receiver are not synchronous in their communication. To test this, the node will store all the request and when available will send all the responses.

Type of test
This test will check the manual and create a test if there is a definition of asynchronous service.

Test Suite case
-Test 1: Create a test with a service that an asynchronous service.

#### 4.4.2.12 The infrastructure should implement some self-healing, self-optimization and self-protection properties.

Name test case
RX.R12

Description
Some common properties a system like this should implement are:

- Self-configuration: Automatic configuration of components
- Self-healing: Automatic discovery, and correction of faults

- Self-optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements.
- Self-protection: Proactive identification and protection form arbitrary attacks

Objectives
Create a test for each self-* properties taking into account that:
- Self-configuration is relational with bootstrapping.
- Self-Healing is relational with robustness, stability and reliability.
- Self-optimization is relational with performance.
- Self-protection is relational with security

Type of test
Some automatic tests will be configured and launched to validate each of the previous self-* properties.

Test Suite case

-Test1: Test that proves the self-configuration property.
-Test2: Test that proves the self-healing property.
-Test3: Test that proves the self-optimization property.
-Test4: Test that proves the self-protection property.

#### 4.4.2.13 The infrastructure should use P2P network architecture – decentralization.

Name test case
RX.R13

Description
P2P systems can be considered to be hybrid or pure systems. In the first case the platform has some nodes that perform administrative functions for the network management and normal nodes. In the case of pure systems, all nodes are equal and distribute these kind of tasks, which means no central nodes are needed.

Objectives
This test will verify the decentralization of the system in terms of configuration and service. In terms of configuration it will verify that a central node doesn't exist. In terms of service, it will be tested whether the service is accessible only for a node or more.

Type of test
This test cannot be done automatically, so it will be done manually.

Test Suite case

- Test_1: For the first item it will be checked in some nodes if their administration server has the same IP that reference to the same machine.

- Test 2: For service point of view, the test will ask if the service provider, offer services only for a node. This node will be disconnected and the service will be used from another one, in order to evaluate this issue.

#### 4.4.2.14 The infrastructure must ensure robustness.

Name test case
RX.R14

Description
Any system must be capable of withstanding errors that should not affect the system stability.

Objectives
The objective of this test is validate the robustness of it, when the service is consumed, the test will simulate errors in different levels, an error could be a failure in the network, a failure in the service or a failure in data.
In order to create this kind of errors, for the case of the network error, the tester will shutdown the node that is consuming a service. To create a service failure the service won't be implemented correctly. Finally, to create a failure in data it will send a wrong type of data.

Type of test
This test will be composed of several tests created to prove the services.

Test Suite case

- Test_1: With this test the data failure will be tested. Data parameters are modified in order to cause errors, in format date, overload integer, string utf-16, etc. The expected response is an error message from the service, but before and after the error the service must work properly.

- Test_2: With this test the service failure will be tested. In order to deploy the service error, another node try find and consume this service, but other service in those nodes will shut down and bootstrap continually in a loop.

- Test_3: With this test the node failure will be tested. While consuming a service, another node will try to find and consume this service. It the nodes are shutdown, the execution of other services mustn't fail.

### 4.4.2.15 The infrastructure must ensure reliability.

Name test case
RX.R15

Description
Reliability implies the ability to recover from an error in the system and return to the previous state before the error occurred. It's the ability of a system or component to perform its required functions under stated conditions for a specified period of time. Some limitations in reliability are due to faults in requirements, design, and implementation.

Objectives
This test must prove that the system is able to recover from failures. Whether caused by system, network or service restart.

Type of test
This test will be composed of several tests created to prove the services.

Test Suite case
From the point of view of the service the following tests will be implemented
- Test_1: After deploying a service, another node will try to find and consume this service, but then the service will be shutdown.
- Test_2: After deploying a service, another node will try to find and consume this service, but then the service will be shutdown and restarted.

From the point of view of the node the following tests will be implemented
- Test_3: After deploying a service, another node will try to find and consume this service, but then the node will be shutdown.
- Test_4: After deploying a service, another node will try to find and consume this service, but then the node will be shutdown and restarted
- Test_5: After deploying a service, another node will try to find and consume this service, but then other nodes will be shutdown and restarted.

#### 4.4.2.16 The system must provide integration with systems that are on different architectures offering interoperability.

Name test case
RX.R16

Description
The requirement *interoperability* is defined as the ability of a system or product that works with other systems or products without special effort.
This test could also imply *portability* as it implies that the application may be available on all machines regardless of the system architecture.

Objectives
The test will prove that the system is able to communicate with another application whatever system is used and also verify that the system can be executed independently of the OS.

Type of test
The test will be done manually.

Test Suite case
- Test_1: Different SO (Windows, Linux and UNIX) and different hardware architectures (Intel, ARM, Motorola) will be tested. The data to be exchanged will be of type IEEE-754 double and quadruple precision, and String in UTf-8 and UFT-16.

#### 4.4.2.17 The infrastructure should be transactional.

Name test case
RX.R17

Description
A transaction is an agreement, communication, or movement carried out between separate entities or objects, often involving the exchange of items of value, such as information, goods, services, and money. Transactional systems are responsible for maintaining consistent data when these are shared with users.
- Atomicity: either all tasks in a transaction are performed, or none of them are.
- Consistency: data is in a consistent state when the transaction begins, and when it ends.
- Isolation: all operations in a transaction are isolated from operations outside the transaction.
- Durability: upon successful completion, the result of the transaction will persist.

Objectives
The main issue of this test will be to evaluate if the system is capable of supporting transactions.

Type of test
This test will be composed of several tests created to prove the requirement.

Test Suite case
Several tests are defined in order to test the concurrent access of readers and writers to the service.

From the point of view of mutual exclusion the following tests will be implemented
- Test 1: While testing starvation, a service will consume the resource of a service more quickly than others, so they don't access the resources. To implement this, a test in loop will consume a service and wait 10 second while another test tries to use this service each second.
- Test 2: While testing deadlock, two or more devices or processes are each awaiting resources assigned to the others, and the computer processing will be evaluated whether is suspended or not.

From the point of view of concurrent access the following tests will be implemented. Regains balance. It's the ability to recover from an error in the system and return to the previous state before the error occurred.
- Test 3: In order to test rollback, it will be checked to go back to previous state when a service fails
- Test 4: In order to test rollback, it will be checked to go back to previous state when the node fails

# 5 STATE OF ART OF TESTING TOOLS

In this chapter are detailed a group of test tools considered in order to evaluate different network architectures previously included on the deliverable. There are many discussions related to different types of testing classification, we have considered unit as non-distributed testing and integration as distributed testing.

*Unit testing* or non-distributed tests are those that run on a single computer system and do not, normally, involve any form of interaction with other computer systems. That means that they include testing a class in isolation of the others. These kind of tests can also be divided into two further categories: local and remote testing.
Local testing involves running test cases on a local computer system without the need to be on a network to run a local test. By comparison remote testing does require that I have a network connection and allow the tester use this connection to run a test on another computer system.

From the other side *integration testing* or distributed testing include testing entire systems made of several classes, several packages and even several external frameworks, such as application servers.
In these cases a test case consists of two or more parts that interact with each other. Each part being processed on a different system. It is the interaction between the different test case components that sets distributed testing apart.

## 5.1 *Tools for unit testing*

First testing tools that are considered for test evaluation, are client applications testing tools. These tools can only be executed on a PC and it's needed to receive feedback information from another node. In this chapter some of them are included:

Junit: It is a simple testing framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. It is a simple, open source framework to write and run repeatable tests. JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test runners for running tests

TestNG: It is a testing framework inspired from JUnit but introducing some new functionalities that make it more powerful and easier to use. It is a testing framework designed to simplify a broad range of testing needs, from Unit testing to Integration testing.
Writing a test is typically a three-step process:

- Write the business logic of your test and insert TestNG annotations in your code.
- Add the information about your test (e.g. the class name, the groups you wish to run, etc...) in a testng.xml file or in build.xml.
- Run TestNG.

## 5.2 *Tools for distributed testing*

These tools communicate with another node sending any message to synchronize remote executions and to establish the correct operation. In this chapter some of them are described:

JSystem:JSystem (http://www.jsystemtest.org/) is an open source framework for writing and running automated system testing. JSystem includes:

- Services Java API - exposes JSystem services
- JSystem Drivers- Java modules used to interfaces with the system under test.
- JRunner - GUI application interface used for creating and running tests scenarios.
- JSystem Agent - Execution engine used to run scenarios on a distributed setup.
- JSystem Eclipse plug-in - accelerates the development environment setup and enforces JSystem conventions. JSystem is based on JUnit (tests and steps) and Ant (execution engine).

Software Testing Automation Framework (STAF):The Software Testing Automation Framework (STAF - http://sourceforge.net/projects/staf ) is an open source, multi-platform, multi-language framework designed around the idea of reusable components, called services (such as process invocation, resource management, logging, and monitoring). STAF removes the tedium of building an automation infrastructure, thus enabling you to focus on building your automation solution.
It is an execution engine which can help you to thoroughly automate the distribution, execution, and results analysis of testcases. STAX builds on top of three existing technologies, STAF, XML, and Python, to place great automation power in the hands of testers. It also provides a powerful GUI monitoring application which allows you to interact with and monitor the progress of your jobs.

Test Environment Toolkit

The Test Environment Toolkit (TET - http://tetworks.opengroup.org ) offers the core facilities of TETware to UNIX and Linux users only. It is provided as an open source, unsupported, command-line product and it is widely used in many test applications.

TETware provides an easy-to-use multi-platform uniform test framework into which local, remote and distributed test suites can be incorporated. It is provided as a supported product for 32-bit Windows, UNIX, and Linux operating systems. This product allows test suites to share a common graphical user interface, promoting sharing of test suites both within an organization and between different organizations. Standardization of the test methodology and tools allows testing efforts to focus away from the harness and tools, increasing efficiency and productivity.

There's a great deal talked about different types of testing: Web testing, regression testing, user testing and of course black box testing, white box testing, even gray box testing. But there is another type of testing that receives less coverage, which is distributed testing. Here we attempt to explain what we consider distributed testing is, and how it compares with non-distributed testing.

.

As it's introduced at the beginning of this chapter non-distributed tests are those that run on a single computer system and do not, normally, involve any form of interaction with other computer systems. They can be divided into two further categories: local and remote testing.

On the other side, distributed testing is different, because a distributed test case consists of two or more parts that interact with each other. Each part should be processed on a different system. It is the interaction between the different test case components that sets distributed testing apart. Typically it is this interaction between different computer systems that is under test.
This is not the same as simultaneous testing. Because even though simultaneous testing involves different test case components being carried out on different processors, and contributing towards a single result, there is no interaction between the test cases or the processors. A further challenge of this tool is that it faces with distributed testing is the type of platform. For example testing a client server application may involve using a Windows client to access one or more UNIX servers, and controlling the whole process from a Linux desktop. So the environment has to be written at a level capable of working across all of these platforms.

## 5.3 *Important testing tools characteristics for distributed P2P scenarios.*

This chapter includes the main characteristics that should be considered in order to select testing tools for distributed peer-to-peer scenarios.

Architecture: The test will be run on a set of different networks. Each of these networks is composed of several elements interrelated with each other. The architecture or test scenario will be adapted to each of the evaluation tests, defined in order to validate each of the requirements.

Network: Network components are considered to be PCs, bridges, routers and the configuration of firewalls and nat. Each of these elements that have to be configured and defined is included in the network definition. In general, the test will be run on a set of different scenarios which could be based of internal local networks or include routers that implement firewall and NAT functions in order to emulate most local networks connected to the internet through them.

Restore scenario: Systems on which tests are going to be evaluated need to launch and run the testing in a clean environment. Therefore, each time the system runs a test, all elements must be restored to its previous state. The test should not leave any trace, and should restore the environment as if it had not implemented any test

Controlled environment: To verify that different releases of the application run in the same environment, each of the defined scenarios should be restored to a clean state each time. It's important to have a clean scenario in order to avoid repeating errors each time we tests a new release. This option is very important for developers and evaluators.

Synchronize [8]: Testing peer-to-peer systems is considered to be so difficult due to the high numbers of heterogeneous and volatile nodes. A test case may be composed of several actions that may be executed on different nodes in the correct order. To ensure the successful behaviour of the test case, a synchronization mechanism is required.
In this kind of systems, actions can be executed in parallel, on different, heterogeneous nodes. Thus, an action can run faster or slower depending on the node computing power. Synchronization is then needed to ensure that a sequence of actions of a test case is correctly executed. Designing a good synchronization mechanism is hard as it must scale up and deal with nodes' autonomy, heterogeneity and volatility.

Administration and configuration: The configuration of both software and hardware equipments should be recorded with all its features in order to avoid implementation phases each time a test should be repeated with a new release. Administration processes are not simple and normally require high administration time, the same happens configuring those scenarios. In order to minimize administrative and configuration delays, virtualization will be used.

Roles [8]: Different kinds of roles are defined in testing tools, which are the coordinator and the testers. *The coordinator*, is responsible for choosing actions that should be executed, and the *testers* are the ones that separately control each node. [10].

The coordinator provides three different interfaces, for action execution, volatility and test case variables:

- register(), ok(), fail(), error(): action registration (performed before all tests) and response for action execution, called by testers once the execution of an action is finished.
- set(), get(): assessors for test case variables.
- leave(), fail(), join(): makes a set of peers leave the system, abnormally quit or join the system.

*Testers* may invoke any public operation available on the node interface as well as make nodes leave and join the system at anytime, according to the needs of the test case.

Distributed systems: This kind of systems is commonly tested using conformance testing. The tester specifies the system using Finite State Machines, Labeled Transition Systems and uses this specification to generate a test suite that is able to verify (total or partially) whether each specified transition is correctly implemented.

The tester then observes the events sent among the different nodes of the system and verifies that the sequence of events corresponds to the state machine. This observation can be achieved using the trace produced by each node.

The integration of the traces of all nodes is used to generate an event time line for the entire system. Distributed systems are particularly difficult to test, the system may have several possible sources of input and output, spread over the network. The nodes that compose the network may be heterogeneous, meaning that the execution time of test case actions varies for each node. Consequently, synchronization among test case actions is necessary. In the particular case of P2P systems, testing is even more complex for two main reasons.

- First, test cases may need to deal with node volatility and simulate the join, departure or failures of nodes at any time.
- Second, test cases must deal with node autonomy and deal with situations where nodes successfully execute several test-cases even though they cannot communicate with each other.

Volatile nodes:[9] A P2P system is composed of a volatile set of nodes, also called peers. Each peer can be a client and a server, as well as a router, since it can route incoming requests to other peers. Peers are autonomous and they can join and leave the system at any time, during the system lifetime. Such dynamic behavior distinguishes P2P systems from classical distributed systems.
A P2P system must be able to work properly even though peers are highly volatile. From the development point of view, a peer is an instance of a P2P application, which executes on a distinct logical node. Programming a peer is a difficult task since it is part of a distributed system, with the classical synchronization issues, and it is programmed with various languages and platforms.

Test Suite [10] A test suite is implemented as a class, which is the main class of the testing application. It contains several test cases, which are implemented as a set of actions and are implemented as annotated methods. Annotations can be attached to methods and to other elements, giving additional information concerning an element: the class is deprecated, a method is redefined, etc. Furthermore, new annotations can be specified by developers.
Method annotations are used to describe the behavior of test case actions: where it should execute, when, in which tester, whether or not the duration should be measured. The annotations are similar to those used by JUnit, although their semantics are not exactly the same.

The choice of using annotations for synchronization and conditional execution was motivated by two main reasons.

- Firstly, to separate the execution control from testing code.
- Secondly, to simplify the deployment of the test cases: all testers receive the same test case. However, testers only execute the actions assigned to them.

The available annotations are listed below:

- *Test*. This is the main annotation, it specifies that the method is actually a test case action. This annotation has four attributes that are used to control its execution: the test case name, the place where it should be executed, its order inside the test case and the execution timeout.
- *Before*. Specifies that the method is executed before each test case. The purpose of this method is to set up a common context for all test cases. The method plays the role of a preamble scenario.
- *After*. Specifies that the method is executed after each test case. Its purpose is to ensure that there will be no interference among different test cases. The method plays the role of a postamble scenario.

## 5.4 *Tools for management*

This chapter includes some management tools that help to generate test environments and the execution of the tests. These tools also provide the right software and configuration for each network elements.

### 5.4.1 Scripting tool

Apache Ant: Ant is a software tool for automating software build processes. It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects ant test suites.

### 5.4.2 Virtualization

A virtual machine system provides a complete system platform which supports the execution of a complete or various operating systems (OS). There are 2 types of virtual software, the one that allows to run a single OS (Workstation) and the one that allow the execution and communication with more than one station (server).
Virtualization is the injection of an abstraction layer between an application and some resources used by that application. It provides a logical rather than physical view of data, computing power, storage capacity, and other resources involving the simulation of combined, fragmented, or resources.

WorkStation

There are a number of companies that provide software that works like a workstation. Named here some of these software

- *Virtual PC*: is a virtualization program for Microsoft Windows operating systems, and an emulation program for Mac OS X on PowerPC-based systems.
- *VMware*: Allows execution of a virtual machine

- *Colinux*: Open Source Linux inside Windows
- *VirtualBox*: is an open source (GPL)/proprietary virtual machine developed by Sun Microsystems

Servers

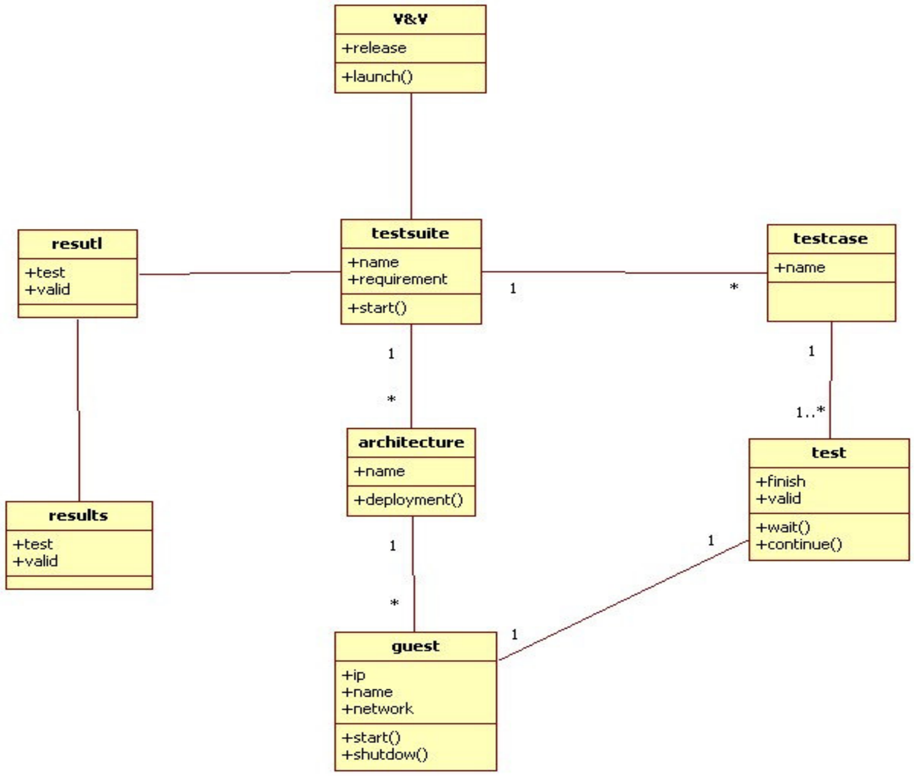
Leading server virtualization software is

- *VMware Server*: is a free, easy-to-use Windows and Linux application that provides a superior introductory experience to server virtualization for businesses who want to optimize use of their technology assets using virtualization.
- *Microsoft Virtual Server*: is a virtualization solution that facilitates the creation of virtual machines on the Windows SO.

Guest virtual machines, allowing communication with host computers share files and even running processes and synchronizing them. Some of them have the ability to change the characteristics of hardware through programming commands and being able to create test environments, settings and restore to original state. The host has the role of coordinator and the guest has the role of testers

### 5.4.3 Management tool

The synchronization of the different elements in the evaluation test is complicated because it must generate events between each of the teams that make up the runtime environment. It could be better to generate a small administration tool that integrates each of these management tools in one. Centralized management of all tests is done in the host and launches the test on computers guest.




The administration tool has requirements that had been definite in chapter 5.3 as Synchronization, control scenario, restore environment, distribute environment, Executes the test suite, it has roles as coordinator and tester and working a virtual scenario.

For each execution of verification and validate test, we has a conjunct of test suit cases. These are executed in an architecture that had been defined as a group of guest machines.

# 6 LAB TESTING ENVIRONMENT

The aim of this testing laboratory is to enable the execution of application tests over the developed P2P platform, Flypeer.

Application tests are those executed by final users and are used to check the performance of a given product, based on checking that each requirement specified on the product specification is satisfied. In this case, the product is aimed to serve as a tool for developing other products, therefore, its final users will be software developers.

It is important to remark that the product under test is a P2P platform, that is, a distributed application. Testing a desktop application could be managed as a simple task, but testing a distributed application, with a P2P architecture, where the application's behavior depends on all the executions of the application distributed on the whole P2P network, involves a higher complexity. Checking every single requirement becomes a hard work, since, it is necessary to build a test environment to validate the application running in a real situation in which different workstations take part interacting among them and testing time is increased by each new distributed node in the network executing the application.

To reduce the difficulty in managing such a complex scenario a non-real controlled environment for testing was built. A controlled environment is a hardware and software scenario where each tests can be run avoiding previous executions to modify the results, so that each test can be reproduced repeatedly to justify the obtained result.

## 6.1 *The Controlled Environment*

The controlled environment built for testing the P2P platform is divided into:

- a hardware environment
- a software environment

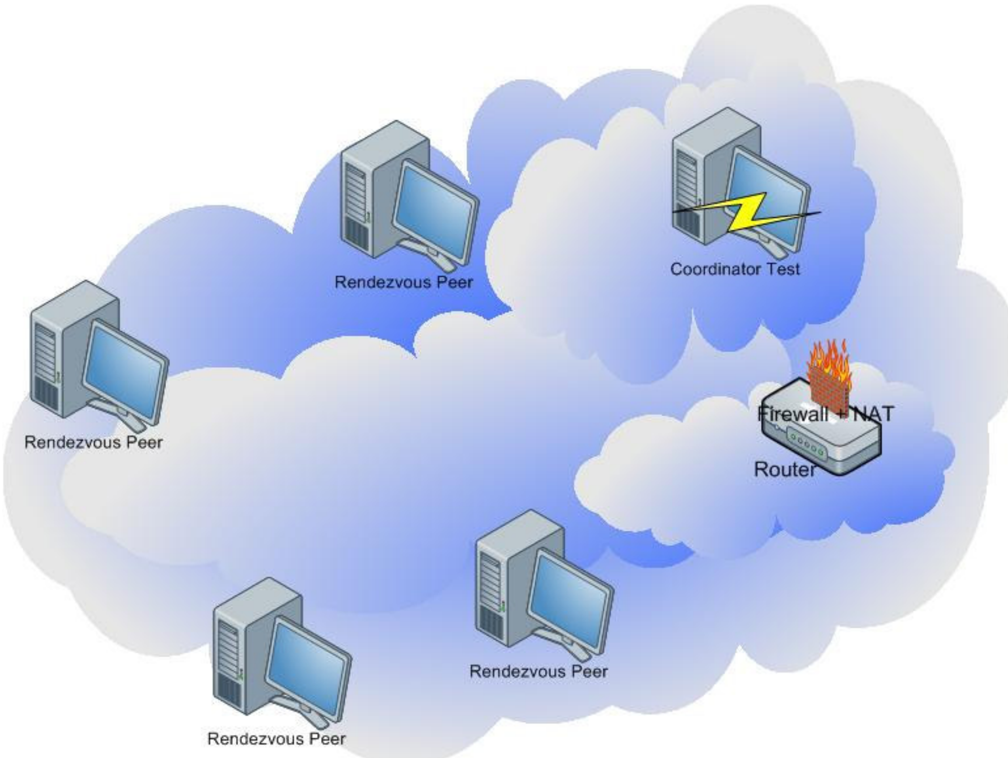



**Figure 16.- Virtualized Lab Testing Environment.**

To establish this hardware environment composed of several workstations, a solution based in virtualized machines was used. By using virtualization, hardware environment is completely defined by a set of properties that can be modified through programming commands. It also makes possible that a copy of the whole testing environment, including both software and virtualized hardware parts, can be easily stored altogether in a certain moment, so that it can be restored after each test execution to provide a clean, non contaminated, environment.

The virtual server used for implementing the laboratory infrastructure runs MVWare software to manage the virtualized machines abstracting hardware complexity and making possible to reconfigure each virtualized machine in an automated way according to the developed tests.

Available instructions in each virtualized node are:

- Turn on
- Turn off
- Configure a network device

## 6.2 *Implementing Test Cases in the Lab Testing Environment*

As already explained in previous sections, a set of Test Suites were described in order to verify a specific requirement each. Each Test Suite involves several test cases which validate, total or partially, the expected behavior.

### 6.2.1 Baseline configuration

In order to implement a test case in the laboratory, it is necessary to define for each test a set of properties to specify the hardware and software properties of the environment.
These set of properties specified for the basic environment configuration are called baseline configuration.

Taken into account the requirements for testing distributed P2P scenarios given in section 5.3 such as volatile nodes, a test case might require that these characteristics are modified for each node during execution time.
Any change in the baseline configuration (adding, modifying or removing some properties) specific for a certain test will be included in the test definition.

The lab management tool, when is preparing the environment for the following test, restores baseline configuration to build a clean environment and ensure that test results are not influenced by previous test executions.

### 6.2.2 Test implementations in each node

Each test is divided into several implementations running in each node of the environment used for the test. To complete the implementation of a test, it is necessary to define previously the role of each node participating in the test and the way it interacts with the test implementations running in other nodes.

### 6.2.3 Test steps

When describing a test, the script goes through a set of states or steps. These states are used by management application to determine the moment to initiate next actions such as the execution of the test implementation on a given workstation or node or finishing the test showing the result.

## 6.3 *Testing tools*

Implementing and executing a test case in the lab testing environment requires the use of a set of tools that configures the environment, initiates applications, registers events and synchronizes actions.

To execute these tasks effectively a set of test elements were created:

- Configuration files
- Several test services
- A coordination tool

### 6.3.1 Configuration files

These configuration files contain the information necessary to configure each node of the lab network, both hardware and software, in order to create the required scenario for the test case.

### 6.3.2 Test services

Test services are services developed using Flypeer platform with testing purposes. Their functionality is determined by the necessities of the test case, p.e. consuming another service.

### 6.3.3 Coordination Tool

The coordination tool does not participate as a test node but it carries out several functionalities, being in charge of configuring each remote test, taking care of both hardware and software.

#### 6.3.3.1 Basic functionality

Given a test definition the coordination tool is in charge of managing the order for execution of test implementations in each node/workstation taking part in the test. Launching a test requires test implementations running in each node where the application is running to be synchronized. Also, a test might be defined depending on the behavior of each node, that is, it might be necessary to execute a test in each node sequentially.

To achieve the synchronization of every single implementation of each test, nodes are managed by the coordination tool.

Test definition also includes any modification that might be necessary to simulate during test execution. Modifications might be necessary in the hardware infrastructure (such as a network failure in a node) or in the software configuration (such as a node with an infinite loop). The coordination tool should interpret these modifications and apply them in the testing environment.

For test execution, the coordination tool launches in each remote machine the necessary applications implemented for each test. It also implements a set of events that will be triggered by processing the trace log files generated by the different test applications running on the test nodes.

When one of these events is triggered, a set of predefined actions will take place. These actions might be to modify the hardware characteristics of a specific machine or to run a command or script in a remote node of the test network.

#### 6.3.3.2 Technologies used for implementation

Two different implementations were carried out for the testing coordination tool:

1) Using script files
   Script files are used to launch simple tests which are easy to execute and validate when no synchronization among tests is required but the interaction of a person is necessary. That means, that a test operator is needed to trigger each test execution step while checking results provided by each node's messages.

2) Using a C# implementation
   This test tool is used for complex tests, since it makes possible to emulate network failures and control test launching by catching events from nodes.

##### *6.3.3.2.1 Scripting*

Ant-apache syntax was used to generate script files using SCP protocol in the coordination tool side and enabling a SSH server in every test node.

A set on management instructions were implemented in the scripts:
- Copy test implementations to each node involved in the test and their corresponding configuration files
- Run test execution in each node and initiation of services to be consumed
- Check availability of JXTA ports to verify the status of Flypeer services (up/down)
- Check service's URL to verify that service is published

When using this implementation, each instruction defined in the test script is executed manually. To know the result of test execution, a remote session using ssh is established to each node to verify log files.

**Pros and Cons**
The implementation of scripts can be completed quickly but, on the other hand, during test execution the operator must control the log files in each node to synchronize events and execution in each node. This task is done manually so test execution is slow.

##### *6.3.3.2.2 Implementation using high-level programming languages*

When it is a requirement that test execution is totally automatic, a total control of the nodes in the test network is mandatory. This accurate control cannot be achieved by using scripts and using high-level programming languages, such as C#, becomes necessary.

In this case, since workstations are virtualized, it is possible to work on them and control both hardware and software parts remotely: physical components, network interfaces, software components, status of launched applications…

The coordination application is executed in one node which is not directly taking part in the test network. This application is in charge of triggering test execution in each node according to the steps defined in test specification. In order to synchronize the test executions running in each node, the coordination application collects trace logs from each node involved in the test to analyze them and launch the corresponding events according to the predefined test sequence.

For the implementation of the test, it was necessary to include a set of libraries that provides communication with VMware interface: VixCOM version 1.6 (later versions of this library do not support used functionalities).
This library is available for languages such as Java, C and C#. One of these languages was chosen for implementing Coordinator application: C#.

**Pros and Cons**

This approach for the Coordinator's implementation has a clear advantage: once a test is defined and implemented, its execution is totally automated, that is no supervision is necessary. Implemented tests can be executed without human interaction and might be scheduled out of work hours when the workstations are not being used.

As a disadvantage, the implementation of this approach implies more work. Since each node generates a considerable amount of log messages, processing these logs and generating the corresponding events is a complex work. During implementation it must be taken into account that the test application running on the test node might not show the expected log message, to cope with that a status message must be included in log output so that Coordinator application can check the application status.

### 6.3.3.3 <u>Application design</u>

#### *6.3.3.3.1 Storage folder structure*

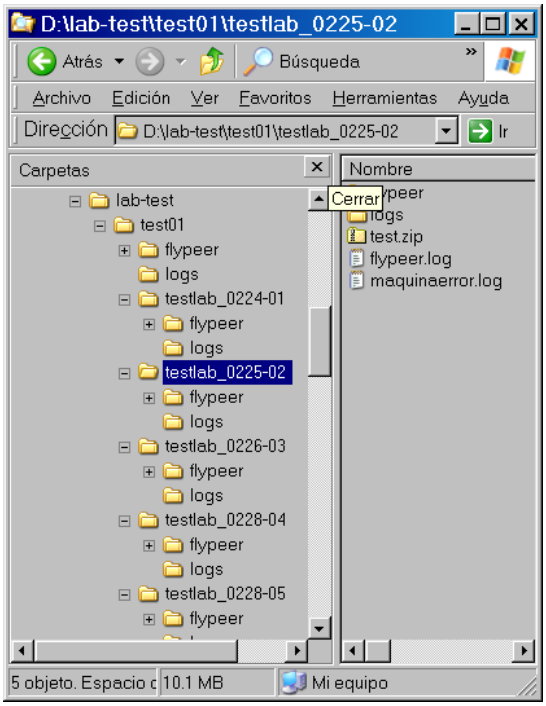



**Figure 17.- Storage folder structure**

There is a folder for each defined test suite where the test implementations for each node taking part in the tests are stored.

The folder structure is created in the node where Coordinator application is running, and contains:

- the test implementations that are going to be transferred to the remote machines
- the corresponding files to configure and launch the test application in the different nodes
- the log files of each node taking part in the tests, which are transferred from each node during test execution

#### *6.3.3.3.2 Events for synchronization*

To let the coordination tool know the status of a test execution in a given node, a notification system based on events has been implemented. The coordination tool implements the actions to be performed when an event is detected.

For example, for the implementation of the Test Suite for the requirement *RX.R01 'The infrastructure must be service-oriented*', in *test_1* '*Service Composer, coordinated',* services must be active previous to start consuming them. To notify the coordination tool that all services to be consumed are up and running, an event is triggered when this situation is detected analyzing the trace log of all nodes. The coordination tool catches this event and runs the next step, that is, it orders another node to start consuming these services.

#### *6.3.3.3.3 Application architecture*

The C# implementation of the coordination tool consists of a class, *"Operation"*, in charge of executing different operations on the remote machines. These operations are:

- to connect to the remote virtualization server where remote machines are generated
- to start virtual machines
- to open remote sessions to virtual machines
- to send/receive files to/from remote virtual machines (executable files, configuration files, log files…)
- to launch test applications in remote virtual machines
- to stop/kill test applications in remote virtual machines

During the execution of tests in a remote node, the application's traces are redirected to log files which are copied periodically (every few seconds) to the node where Coordinator application is running.

A *'Suitcase'* class is defined for each machine or remote node where the test suitcase is going to be executed. *'Suitcase'* class contains:

- a list of *'Machine´* classes which contain information of the remote machine such as the machine's name and IP address.
- a list of *'Test´* classes which refer to the tests' implementations

*'LabStarting'* class initializes the laboratory environment to execute tests. This class is in charge of copying executable files and libraries into the remote machines and of configuring them for a test execution.

*'LabRunning'* class is in charge of executing the test implementation in the corresponding node. It is also in charge of transferring periodically, according to a predefined interval, the log files generated in the remote node to the Coordinator node.

*'ReadStreams'* class is in charge of parsing in the Coordinator node the log files received from each node. This class is also in charge of triggering the events previously defined for each file.

To enable *'ReadStreams'* class for throwing events, it is necessary to implement the interface called *'EventFile'*. This interface defines 2 operations: *addEvent* and *executeEvent* (which implements the actions to be done when the event is triggered).

It is defined a class that is an interface for each of the files generated within the application. Standard and error output of test applications running in test nodes are redirected to two files and the interface *'EventFile'* is implemented for each. Implemented classes are *'EventFile1'* and *'EventFile2'*.

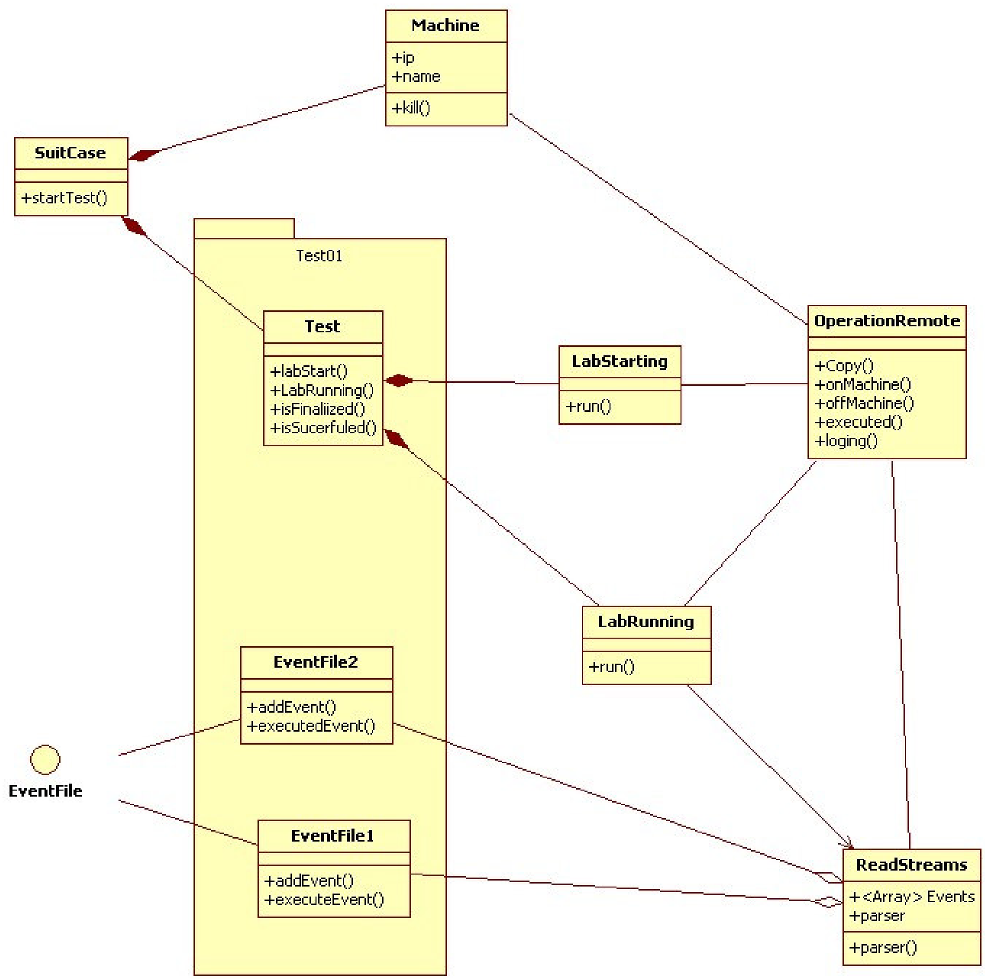



**Figure 18.- Class Diagram for Coordination Tool**

## 6.4 *Executing Test Cases*

For the execution of a specific test, first step is to restore baseline network configuration. Then the test implementation is copied into each node or workstation taking part in the test. This test implementation may not be the same for each involved node, depending on the role they play.

Finally, the network is configured according to the specific characteristics defined by the test specifications for each of the nodes and network elements.

Once the environment is configured and all software components are copied into the corresponding nodes, the coordination tool starts executing the different steps described in test definition.

### 6.4.1.1 Example: Test Suite case RX.R01

An example for executing a test is explained for Test Suite case RX.R01 that validates requirement '*The infrastructure must be service-oriented*'.

This test suite is formed by 3 tests:

1) *Service Composer coordinated*
2) *long-running transaction*
3) *loosely-coupled*

#### *6.4.1.1.1 Deploying the test in the lab testing environment*

The *Service Composer coordinated* test requires the execution of 9 services distributed among 3 different nodes (#1,#2 and #3) while another node (#4) will be consuming these 9 services.

Test specification specifies:

- necessary number of nodes for text execution. In this case: 4
- node's functionality:
    - 3 nodes (#1,#2 and #3) publish 3 services each
    - 1 node (#4) consumes services, once they are properly published

To execute properly the test, the coordination tool will first

- restore baseline configuration
- configure each node and any other element in the laboratory network according to test specification
- copy in all nodes the test application part corresponding to each role

### *6.4.1.1.2 Sequence Diagram*

The figure below shows the sequence diagram for the example. In this test case, there are 3 entities interacting: one is acting as the coordination tool and the other two are acting as nodes where the test applications are run.

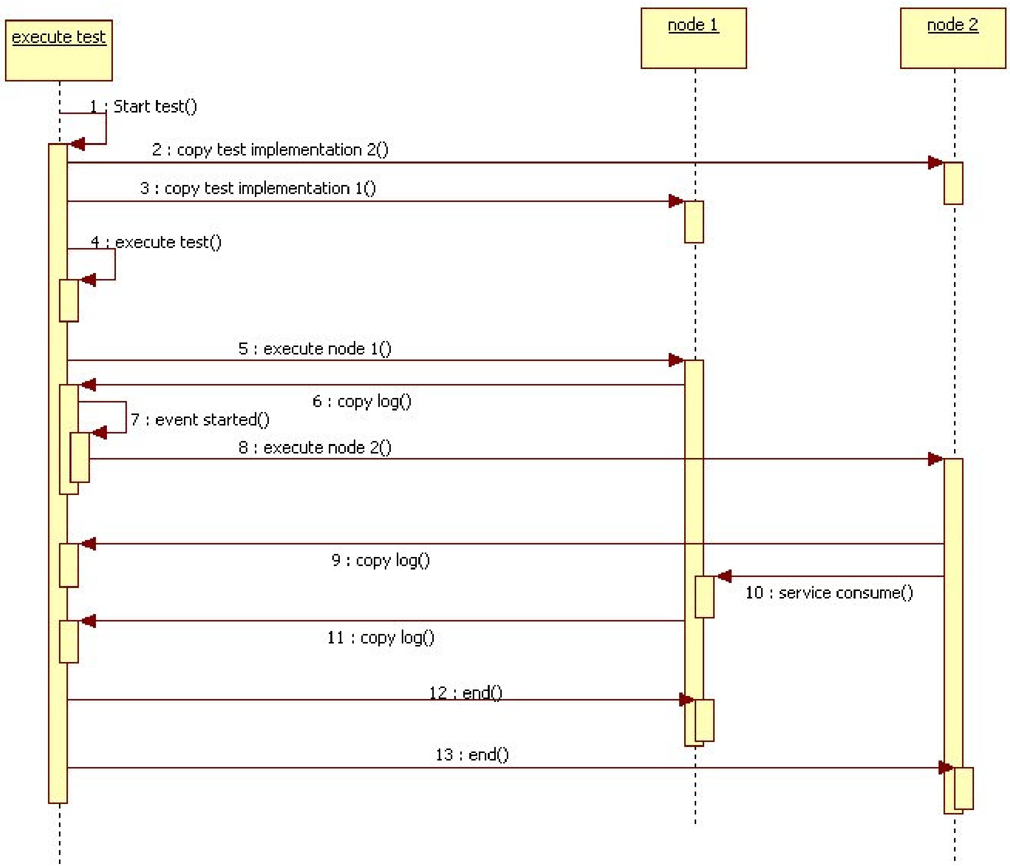



**Figure 19.- Sequence Diagram**

This basic test verifies the test case in which one of the test nodes (node #1) publishes a service and the other (node #2) consumes it. In order to run the test exactly as the expected scenario, the coordination tool application first executes the test implementation in node #1 and then, after verifying that the service is deployed and published, the test implementation in node #2 is started so that it consumes the published service in node #1.

Previous to the start of test execution, the first step is to transfer into each test node the implementation of the test specific for each node.

The coordination application will then initiate remotely the execution of test application in node #1 and wait for the node #1 to notify that the test is ready. This notification is detected by the coordination tool when processing the log file: the coordination application triggers an event when it detects in log file that node #1 has registered that the service has been published. Both log files, from node #1 and node #2, are collected by the coordination tool periodically.

Next, the coordination tool will remotely initiate the execution of test implementation in node #1 and waits till it detects in the trace log that the services has been published and the test execution should go one step further.

After detecting that node #1 has notified that the service is published, the coordination tool will remotely execute the test implementation in node #2, which will run a search for the service in node #1 and consume it.

The coordination tool will detect in log files that the service is consumed (or not) and will determine that the test has come to an end so it will remotely finish the execution of test implementation in both nodes and process the result of the test.

# 7 ROADMAP.

## 7.1 *Releases Roadmap.*

Next table summarized the Project Roadmap the project development team established by the end of September.

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| **Month** | **End of Sep** | **End of Oct** | **End of Nov** | **End of Dec** | **End of Jan** |
| **Flypeer** | Transaction Support | Lock Model (IDG) | Lock Model(EDG) | Easier Development Model | Stable Release |
| **Identity** | JXTA Integration<br>Authentication | Identity Verification-(Requirements and Scenarios)<br>(Non-testable) | Identity Verification | | |
| **Trust** | Standalone Demo | | Trust manager and Overlay | ID Trust Algorithms | |
| **Semantic Search** | JXTA based Demo | Guigoh Integration | Flypeer Integration | Trust Integration | Scalability solutions |
| **Visualisation** | | Willie Framew ork 2.1 | | | |
| **Service Container (Willie)** | | Willie Framew ork 2.1 | | Best practice for Implementing Flypeers Services | |

**Table 20.- Roadmap of Releases.**

Things to be concreted:

- Detailed description of the functionalities obtained for each module.
- Are this functionalities under development or they should be considered finished with each of the releases?

### 7.1.1 Release R0 Functionalities Description.

Although R0 was not a planed release of the development team we, the test team, considered it in order to start to development the test plan and to create the information flow by iteration with the objective to obtain the functionalities of R2, R3, R4 and R5 in advance. This will allow to both, the development and the test staff , to define the test, schedule it together with the bug tracking.

We do not include the R1 release in the plan as when the Roadmap was established and this document was first released the detailed description of the R1 functionalities were not yet available. So we considered it was a second iteration of the testing loop.

| **Release:** | **R0** | **Requirements Affected.** |
|---|---|---|
| **Month** | **End of Aug** | **Unknown** |
| **Module** | **Functionalities High Level Description** | **Functionalities Detailed Description.** |
| **Flypeer** | **Unknown** | **Unknown** |
| **Identity** | **Unknown** | **Unknown** |
| **Trust** | **Unknown** | **Unknown** |
| **Semantic Search** | **Unknown** | **Unknown** |

### 7.1.2 Release R1 Functionalities Description.

| Release: | R1 | Requirements Affected. |
|---|---|---|
| Month | End of Sep | |
| Module | Functionalities High Level Description | Functionalities Detailed Description. |
| Flypeer | Transaction Support | |
| Identity | JXTA Integration Authentication | |
| Trust | Standalone Demo | |
| Semantic Search | JXTA based Demo | |
| Service Container (Willie) | | N. A. |

### 7.1.3 Release R2 Functionalities Description.

| Release: | R2 | Requirements Affected. |
|---|---|---|
| Month | End of Oct | |
| Module | Functionalities High Level Description | Functionalities Detailed Description. |
| Flypeer | Lock Model (IDG) | |
| Identity | Identity Verification-(Requirements and Scenarios) (Non-testable) | |
| Trust | | N. A. |
| Semantic Search | Guigoh Integration | N.A. ita test should test de P2P infrastructure. |

### 7.1.4 Release R3 Functionalities Description.

| Release: | R3 | Requirements Affected. |
|---|---|---|
| Month | End of Nov | |
| Module | Functionalities High Level Description | Functionalities Detailed Description. |
| Flypeer | Lock Model(EDG) | |
| Identity | Identity Verification | |
| Trust | Trust manager and Overlay | |
| Semantic Search | Flypeer Integration | |

### 7.1.5 Release R4 Functionalities Description.

| Release: | R4 | Requirements Affected. |
|---|---|---|
| Month | End of Dec | |
| Module | Functionalities High Level Description | Functionalities Detailed Description. |
| Flypeer | Easier Development Model | |
| Identity | | N. A. |

| | | |
|---|---|---|
| **Trust** | ID Trust Algorithms | |
| **Semantic Search** | Trust Integration | |

### 7.1.6 Release R5 Functionalities Description.

| **Release:** | **R5** | **Requirements Affected.** |
|---|---|---|
| **Month** | **End of Jan** | |
| **Module** | **Functionalities High Level Description** | **Functionalities Detailed Description.** |
| **Flypeer** | Stable Release | |
| **Identity** | | **N. A.** |
| **Trust** | | **N. A.** |
| **Semantic Search** | Scalability solutions | |

# 8 MANUAL PERFORMED TESTS FOR FLYPEER.

The following section includes the results of tests done on the different modules of Flypeer.

## 8.1 *Test of the version 0.3.1 of the Flypeer Module (R0).*

### 8.1.1 Introduction.

Due to the lack of documentation the correct definition of the Test Roadmap could not be done and was impossible to perform beta test. In order to do some work related the evaluation was done following documentation contained in D1, D2 and D3 documents.

### 8.1.2 Environment Description.

| Environment Description. | |
|---|---|
| **Date of the Report:** | 20/09/2009 |
| **Module/Platform Version:** | Flypeer 0.3.1 (Released on 20080828) |
| **Documentation:** | There are 3 manual version<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |

### 8.1.3 Performed Tests.

Documentation included on D1, D2 and D3 documents was followed in order to validate this module.

Next table summarizes the tests we ran together with the id of the reported errors more detailed can be obtained in Appendix II.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R0.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R0.D.01 | 30/09/2009 | |
| | R0.D.02 | 30/09/2009 | |
| | R0.D.03 | 30/09/2009 | |
| | R0.S.04 | 30/09/2009 | |
| | R0.S.05 | 30/09/2009 | |
| | R0.S.06 | 30/09/2009 | |
| | R0.S.07 | 30/09/2009 | |
| | R0.S.08 | 30/09/2009 | |

**Table 21.- Evaluation of the Flypeer 0.3.1 version.**

## 8.2 *Test of the version 0.5 of the Flypeer Module (R2).*

### 8.2.1 Introduction.

We test the new release Manuals. There is very little documentation to test the product depth. The test is to create a node and publish a service "*HelloWorld*".

### 8.2.2 Environment Description.

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 18/12/2009 |
| **Module/Platform Version:** | Flypeer 0.5. (Released on 20091218) |
| **Documentation:** | There are 3 manual versión<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>    • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>    • The more complete manual as it includes references to the source code to consume a service. |
| **Others:** | |

### 8.2.3 Performed Tests.

Although we did not run any test and because we did no received any new about the state of the errors of the previous release we reproduce here the same list of errors.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R2.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R2.D0001 | 18/12/2009 | No |
| | R2.D0002 | 18/12/2009 | No |
| | R2.D0003 | 18/12/2009 | Yes |
| | R2.D0004 | 18/12/2009 | No |
| | R2.D0005 | 18/12/2009 | Yes |
| | R2.D0006 | 18/12/2009 | No |
| | R2.D0007 | 18/12/2009 | No |
| | R2.S0008 | 18/12/2009 | No |

**Table 22.- Evaluation of the Flypeer 0.3.1 version.**

## 8.2.4 Evaluation of the Platform against its Requirements.

| | | | | |
|---|---|---|---|---|
| | **Date** | **20/09/2009** | | |
| | **Evaluated Release** | **Flypeer 0.3.1** | | |
| **Rid** | **R1 Requeriments: the infrastructure must...** | **State** | **Implementation Release** | **Notes** |
| **R1.1** | Be service oriented. | Unknownn | Unknown | |
| **R1.2** | Manage service deployment. | Unknown | Unknown | |
| **R1.3** | Manage service searching. | Unknown | Unknown | |
| **R1.4** | Manage service consuming. | Unknown | Unknown | |
| **R1.5** | Allow service composition | Unknown | Unknown | |
| **R1.6** | Manage security in the transport layer. | Unknown | Unknown | |
| **R1.7** | Manage identity. | Unknown | Unknown | |
| **R1.8** | Make bootstrapping connection easier. | Unknown | Unknown | |
| **R1.9** | Manage transactions in service composition. | Unknown | Unknown | |
| **R1.10** | Give support for scalability and performance. | Unknown | Unknown | |
| **Rid** | **R2 Requeriments: the infrastructure should...** | | | |
| **R2.1** | Be based on web services standards. | Unknown | Unknown | |
| **R2.2** | Give support for semantic management. | Unknown | Unknown | |
| **R2.3** | Run on different software platforms. | Unknown | Unknown | |
| **R2.4** | Run on different hardware platforms. | Unknown | Unknown | |
| **R2.5** | Be integrated to other infrastructures. | Unknown | Unknown | |
| **R2.6** | Be based on JBI and OSGi standards. | Unknown | Unknown | |
| **R2.7** | Support for asynchronous communication. | Unknown | Unknown | |
| **R2.8** | Support for extended web-services protocols. | Unknown | Unknown | |
| **R2.9** | Manage distributed identity. | Unknown | Unknown | |
| **R2.10** | Implement some self-* properties. | Unknown | Unknown | |
| **R2.11** | Use P2P network architecture. | Unknown | Unknown | |
| **R1.12** | Give support for semantic management. | Unknown | Unknown | |
| **Rid** | **R2 Requeriments: that can't be implemented into the infrastructure.** | | | |
| **R3.1** | Legal Regulatory Framework. | Not Valuable | Unknown | |
| **R3.2** | Quick Bootstrapping. | Unknown | Unknown | |
| **R3.3** | Have a support community. | Not Valuable | Unknown | |
| **R3.4** | Have an Integrated Development Environment. | Unknown | Unknown | |
| | **General Evaluation:** | **The platform does not fullfill any requirement and we do not know which and when there are going to be, so it is not possible to create a test plan.** | | |

**Table 23.- Evaluation of the Flypeer 0.3.1 version.**

## 8.3 *Test of the version 0.6.0 and version 0.6.1 of the Flypeer Module (R3).*

### 8.3.1 Introduction.

First, we downloaded the examples as Agency Service, Hotel service or Flight booking service from the mercurial repository.

### 8.3.2 Environment Description.

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 26/02/2010 |
| **Module/Platform Version:** | Flypeer 0.6.0 (Released on 20100112)<br>Flypeer 0.6.1 (Released on 20010114) |
| **Documentation:** | There are 3 manual version<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>    • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>    • The more complete manual as it includes references to the source code to consume a service. |
| **Service sum** | We received by email from deniswsrosa@gmail.com a service call sum at 03/02/2010 |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show (D4) |

### 8.3.3 Performed Tests.

Documentation included on D1, D2 and D3 documents was followed in order to validate this module.

Next table summarizes the test we ran together with the id of the reported errors more detailed can be obtained in Appendix II.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R3.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R3.D0001 | 12/01/2010 | NO |
| | R3.D0002 | 12/01/2010 | No |
| | R3.S0003 | 12/01/2010 | Yes |
| | R3.D0004 | 18/01/2010 | No |
| | R3.D0005 | 18/01/2010 | No |
| | R3.S0006 | 18/01/2010 | Yes |

**Table 24.- Evaluation of the Flypeer 0.3.1 version.**

## 8.4 *Test of the version 0.7.0 of the Flypeer Module (R4).*

### 8.4.1 Introduction.

We test the **service sum.** The module Sum has two components: a service client and service server. This service is executed in local and remote configuration. It's simple, we only had to execute 2 files of type JAR.

### 8.4.2 Environment Description.

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 26/02/2010 |
| **Module/Platform Version:** | Flypeer 0.7.RC1 (Released on 20100211 |
| **Documentation:** | There are 3 manual versions<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Service sum** | We received by email from deniswsrosa@gmail.com a service call sum at 03/02/2010 |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show (D4) |

### 8.4.3 Performed Tests.

Documentation included on D1, D2 and D3 documents was followed in order to validate this module.

Next table summarizes the test we ran together with the id of the reported errors more detailed can be obtained in Appendix II.

| **Test Id** | **Description** | **We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.** | |
|---|---|---|---|
| **R3.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R4.S0001 | 26/02/2010 | yes |
| | R4.S0002 | 26/02/2010 | Yes |
| | R4.S0003 | 26/02/2010 | Yes |
| | R4.s0004 | 26/02/2010 | Yes |

**Table 25.- Evaluation of the Flypeer 0.7.0 version.**

## 8.5 *Test of the version 0.8 of the Flypeer Module (R5).*

### 8.5.1 Introduction.

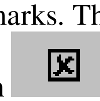
Because there is no requirements document, we started testing the requirements defined in this document, without knowing whether these requirements are implemented.

### 8.5.2 Environment Description.

On the Flypeer web page there is a new document “Javadoc”. Javadoc is a documentation generator from Sun Microsystems for generating API documentation in HTML format from Java source code. The "doc comments" format used by Javadoc is the de facto industry standard for documenting Java classes.

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 20/04/2010 |
| **Module/Platform Version:** | Flypeer 0.8. (Released on 20100407) |
| **Documentation:** | There are 3 manual versión<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service.<br>There are JavaDoc Documentation:<br>• http://kenai.com/projects/flypeer/downloads/download/flypeer-0.8-javadoc.tar.gz(D5) |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show (D4) |

### 8.5.3 Performed Tests.

To mark the results of a test are 3 type of marks. The tests that pass are marked with  and the tests that fail are marked with  and mark with a “dot” when they can’t be tested
Only, when all subtests fail, then the test is marked as fail, otherwise the tests are marked as passing.
We implemented only the test suit case RX.R001 defined in this document. The others test are evaluated manually.

- ✓ The test cases RX.R001 are composing by subtest *Service composer, coordinator,* subtest *Long-running transaction* and subtest: *Loosely-coupled.*

☒ The test case RX.R002 can't implement because the framework hasn't the infrastructure manager to deploy services.

☒ The test case RX.R003 can't be implemented because the framework hasn't a full documentation to create a search of service.

✓ The test case RX.R004 implements the static composition but the framework doesn't implement dynamic composition.

☒ The test case RX.R005 can't be implemented because there is no unauthorized access control. When a service is deployed every body can used

✓ The test case RX.R006 some information/data, belonging to the user (login and password) is send encrypted.

✓ The test RX.R007, the infrastructure must make bootstrapping connection easier, the framework can connect to the networks. The frameworks only connected to the networks when all parameters are configured.

✓ The test RX.R008, the infrastructure must give support for scalability and performance.

✓ The test RX.R009, the infrastructure should be based on standards. The framework uses a few standards only in definition of parameters, but the frameworks doesn't use standards in the identity or in the methods used to consume the services.

✓ The test RX.R010, the infrastructure should run on different software and hardware platforms- portability. The infrastructure made in Java and Java support this feature.

✓ The test RX.R011, the infrastructure should have support for asynchronous communication. The asynchronous communication is support for the framework. Only in service consume client, when the service provider are not support asynchronous.

✓ The test RX.R012, the infrastructure should implement some self-healing, self-optimization or self-protection properties. The infrastructure implements self-discovery with other nodes.

✓ The test RX.R013, the infrastructure should use p2p network architecture decentralization. The infrastructure implements an architecture based on p2p.

✓ The test RX.R014, the infrastructure must ensure robustness. The infrastructure is not robust, because the errors affect the system stability.

☒ The test RX.R015, the infrastructure must ensure reliability. The ability to recover from an error in the system and return to the previous state before the error occurred.

✓ The test RX.R016, the system must provide integration with systems that are on different architectures offering interoperability. The infrastructure supports all systems that support java.

▪ The test RX.R017, the infrastructure should be transactional. In the documentation said: Not implemented a classic definition of transactions, which does not apply this requirement

### 8.5.4 Evaluation of the Platform against its Requirements.

Next table summarizes the evaluation of Flypeer against the Requirements. The evaluation is based in four possible states which are classified by the comments which appear on the column situated on the right of the table. They should have be interpreted as:

- Unknown: it has not been possible to know if Flypeer implements the Requirements.
- Not implemented: from the documentation or as result of a test our conclusion is that the requirements is not fulfilled.
- Partially Implemented: the requirements is either partially fulfilled or it is not correctly documented.
- Implemented: the requirements are fulfilled by the P2P infrastructure.

If as the SMEs in their evaluation of the platform, we are asked to give the platforms a grade, it would be 6 in a scale from 0 to 10. Our decision is based on:

- The lack of documentation which can make the platform sustainable after the finishing of the project.
- It has been difficult to develop both the test cases and the pilot project we developed together with the SMEs. The reading of this comment and the previous one should take into consideration that when we performed the tests the platform was under deployment and in consequence it was very hard to keep consistency between developments and documentation and the stability of the code for the development team.
- Although many tools has been developed to be used with in Flypeer and to make it more easy to use there is a lack of convergence between them.
- The basic infrastructure seems to work and the high level services and the surrounding tools and researches can be used by the Opaals partners to create new releases in the future.

From our point of view, as an R+D+i center which works closely with the enterprises for future projects it would be a good idea to follow a project

methodology: defining the desired results, defining interfaces, task and milestones where the project can be reviewed and the outputs redefined. This will allow obtaining a set of results which can be closed to the Enterprises demands.

| | | Date | 30/05/2010 | | |
|---|---|---|---|---|---|
| | | Evaluated Release | Flypeer 0.8 | | |
| Rid | | R1 Requeriments: the infrastructure must… | TestPlafaform | Implementation Release | Comments |
| R1.1 | RX.001 | Be service oriented. | Implemented | Unknown | |
| R1.2 | RX.002 | Manage service deployment. | Not Implemented | Unknown | It has to be hardcoded |
| R1.3 | RX.003 | Manage service searching. | Unknown | Unknown | There is not enough documentaton. |
| R1.4 | | Manage service consuming. | Unknown | Unknown | There is not enough documentaton. |
| R1.5 | RX.004 | Allow service composition | Partially Implemented | Unknown | It is based on Transactions but it does not work right in distributed nodes. |
| R1.6 | RX.005 | Manage security in the transport layer. | Partially Implemented | Unknown | There is not a clear transport layer definition. It is supported by JXTA but it is not documented how to configurate it. |
| R1.7 | RX006 | Manage identity. | Implemented | Unknown | Some information data, belong the user (login and password) is send encrypting. |
| R1.8 | RX.007 | Make bootstrapping connection easier. | Partially Implemented | Unknown | There is not enough documentaton. |
| R1.9 | RX.017 | Manage transactions in service composition. | Unknown | Unknown | It seems to work right in a local node but we were not able to make it runs in distributed scenarios. |
| R1.10 | RX.008 | Give support for scalability and performance. | Partially Implemented | Unknown | JXTA supports it but we were not able to test its performance. |
| Rid | | R2 Requeriments: the infrastructure should… | | | |
| R2.1 | RX.009 | Be based on web services standards. | Partially Implemented | Unknown | It allow using WSDL as interface for existing services but, for example, messages are binnary Java objects. |
| R2.2 | | Give support for semantic management. | Unknown | Unknown | There is not enough documentaton. |
| R2.3 | RX.010 | Run on different software platforms. | Implemented | Unknown | Java is a multiplatform runtime framework. |
| R2.4 | RX.010 | Run on different hardware platforms. | Implemented | Unknown | Java is a multiplatform runtime framework. |
| R2.5 | Rx.016 | Be integrated to other infrastructures. | Patilly Implemented | Unknown | It just allow the integration with legacy system through WSDL interface but there is not an standard "service interface" to integrated with other service plataforms. |
| R2.6 | | Be based on JBI and OSGi standards. | Not Implemented | Unknown | |
| R2.7 | RX.011 | Support for asynchronous communication. | Implemented | Unknown | |
| R2.8 | | Support for extended web-services protocols. | Not Implemented | Unknown | |
| R2.9 | | Manage distributed identity. | Not Implemented | Unknown | |
| R2.10 | RX.012 | Implemented some self-* properties. | Partially Implemented | Unknown | JXTA provides ways to allow different kinds of nodes to changes their role depending on the state of the network, but it is not documented. |
| R2.11 | RX.013 | Use P2P network architecture. | Implemented | Unknown | It is provided by JXTA. |
| R2.12 | | Give support for semantic management. | Partially Implemented | Unknown | Some Flypeer tools allows to sematically annotated and search information (f.e WSDL files) but it is not documented how it can be integrated within Flypeer network. |
| | RX.014 | infrastructure should ensure robustness | Patially Implemented | Unknown | |
| | RX.015 | the infrastructure should ensure reliability | UnImplemented | Unknown | |
| Rid | | R2 Requeriments: that can't be Implementeded into the infrastructure. | | | |
| R3.1 | | Legal Regulatory Framework. | Not Valuable | Unknown | |
| R3.2 | | Quick Bootstrapping. | Not Implemented | Unknown | There is not a complete documentation about the Network seeding. |
| R3.3 | | Have a support community. | Not Valuable | Unknown | |
| R3.4 | | Have an Integrated Development Environment. | Partially Implemented | Unknown | Some Flypeer Tools has been developed but they are not integrated in a common user interface and they can not be used to develop or integrate services within Flypeer. |
| | General Evaluation: | The platform does not fullfill any requirement and we do not know which and when there are going to be, so it is not possible to create a test plan. | | | |

**Table 25.- Final Evaluation Report.**

# 9 FINAL EVALUATION REPORT AND RECOMMENDATIONS FROM BETA-TESTING

In order to validate an application, it's important to consider documentation related like how it's designed, its implementation, the definition of requirements, how it's being tested by the developer or simply a user manual. Due to the lack of information some of the requirements have to be defined as general requirements that probably don't suit with this application and most of the tests have been done without the desired level of depth.

However the application is well designed, it's based on Java, which means it's based on an independent technology. Also it is based on open standards by using the JXTA technology that supports P2P architecture, which gives the application functionalities like robustness, scalability, load balancing, self-organization, etc.

Versions 0.3 and 0.7 have been tested as alpha version, where as version 0.8 has been considered the beta version. It would have been better if the application could have been done on time in order to validate all developed functionalities.
Probably the bigger problem for the evaluation has been the lack of documentation. Most of the time spent in the test phase, has been in configuring the platform and implementing the services, which we could have avoid using a correct user manual.

# Appendix I. REFERENCES.

[1] Brendon J. Wilson, "JXTA", 2002-http://www.brendonwilson.com/projects/jxta
[2] D5.3 Re-engineered Servent - OPAALS Project (Contract n° FP6-034824)
[3] Kaarthik Sivashaummigam, "Project JXTA"
http://lsdis.cs.uga.edu/~kaarthik/SemEnt/Project%20JXTA.ppt
[4] JXTA Community projects, https://jxta.dev.java.net/?listName=discuss
[5] D3.2 Report on formal analysis of autopoietic P2P network, together with predictions of performance - OPAALS Project (Contract n° FP6-034824)
[6] D3.1 Preliminary architecture for P2P network focusing on hierarchical virtual super-peers, birth and growth models - OPAALS Project (Contract n° FP6-034824)
[7] D3.6 Consensus detailed architecture of the OPAALS DE - OPAALS Project (Contract n° FP6-034824)
[8] E.Almeida, G.Sunye P.Valduriez. Action Synchronization in P2P System Testing
[9] E.Almeida, G.Sunye P.Valduriez. Testing Peers' Volatility
[10] E.Almeida, G.Sunye P.Valduriez. A Framework for Testing Peer-to-Peer Systems
[11] ISO/IEC 27001
[12] The 'European Interoperability Framework for pan-European eGovernment Services' http://ec.europa.eu/idabc/servlets/Doc?id=19528
[13] Halepovic, E.; Deters, R.; "JXTA performance study", Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference
[14] http://www.ibm.com/
[15] http://www.wikipedia.com
[16] Scott Oaks, Bernard Traversat, Li Gong , author of O'Reilly's "JXTA in a Nutshell"
[17] Deliverable D3.1: Preliminary architecture for P2P network focusing on hierarchical virtual super-peers, birth and growth models
[18] Improvement of JXTA Protocols for Supporting Reliable Distributed Applications in P2P System

# Appendix II. R0 Details of Tests Performed.

## A.II.1 Environment Description.

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 20/09/2009 |
| **Module/Platform Version:** | Flypeer 0.3.1 (Released on 20090828) |
| **Documentation:** | There are 3 manual versions<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Others:** | |

## A.II.2 Test R0.0001

### A.II.2.2 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.
We considered next network topology in order to run the test:

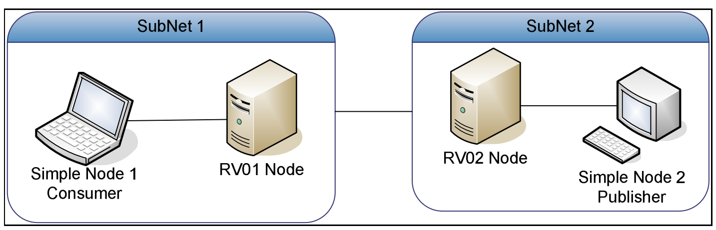



**Figure 26.- Network Topology**

We next describe the previous diagram shortly:

- 2 Rendezvous nodes are deployed (RV01 and RV02) under different subnets.
  - Error in documentation: there is not any indication that in the beginning of a JXTA network each of the RV nodes has to refer/be "linked" each other.
- A simple node (Simple Node 1) is deployed into the Subnet1 and it is "linked" to RV01 node. This node is going to consume the "*Helloword"* service available in the Simple Node 2.
- Another simple node (Simple Node 2) is deployed in the other subnet an linked to the local Rendezvous (RV02). This node published a service "*HelloWorld*" service.

### A.II.2.3 Documentation Errors.

| Document | Error id | Description |
|---|---|---|
| **D1, D2** | R0.D.0001 | The flypeer*.jar file name does not correspond to the one reflected at the command line. |
| **All** | R0.D.0002 | There is not any suggestion about the necessity of connecting a simple node to a Rendezvous one before the first is able to consume a service. In order to consume a service a node has to implement the *PeerStatusNotifier* class. |
| **All** | R0.D.0003 | There is not an explanation about the PeerStatusNotifier its purpose and about |

<table>
<tr><td></td><td></td><td>the necessity of a node to intializate it in order to send a request.<br>public class MyNotifier implements PeerStatusNotifier {<br>public void peerStarted() {<br>System.out.println( "Peer Started!" );<br>}</td></tr>
</table>

# A.II.2.4 Software Errors.

<table>
<tr><th>Module</th><th>Error id</th><th>Description</th></tr>
<tr><td>Flypeer</td><td>R0.S.0004</td><td>When the remote service is invoked from a client, the first call run right, the second fails and the third one fail on so on.<br>The next are some lines recovered from the log. The next is the information provided by the log when the service call work right:<br>INFO: JXME Proxy Service started.<br>20-sep-2009 13:31:31 net.jxta.impl.endpoint.relay.RelayClient startClient<br>INFO: Started client : relay://uuid-59616261646162614E504720503250334D59504545524200800080003<br>20-sep-2009 13:31:31 net.jxta.impl.endpoint.relay.RelayClient run<br>INFO: Start relay client thread<br>20-sep-2009 13:31:31 net.jxta.impl.endpoint.relay.RelayTransport startApp<br>INFO: Relay Message Transport started<br><br>The next sentences are printed when the service call does not work right:<br><br>INFO: JXME Proxy Service started.<br>10-sep-2009 13:30:28 net.jxta.impl.endpoint.relay.RelayClient startClient<br>INFO: Started client : relay://uuid-59616261646162614E504720503250334D59504545524200800080003<br>20-sep-2009 13:30:28 net.jxta.impl.endpoint.relay.RelayTransport startApp<br>INFO: Relay Message Transport started<br>20-sep-2009 13:30:28 net.jxta.impl.endpoint.relay.RelayClient run<br>INFO: Start relay client thread</td></tr>
<tr><td>Flypeer</td><td>R0.S.0005</td><td>Sometimes when a service invocation is being performed it is logged a message saying that the recipient port had been previously closed.<br>The sequence of calls to the service are:<br>• First call to the service<br>14-sep-2009 14:23:30 net.jxta.impl.endpoint.tcp.TcpMessenger <init><br>INFO: Creating new TCP Connection to : tcp://193.144.228.57:9765 / 193.144.228.57:9765<br>A lot of mesagge ….<br>14-sep-2009 14:23:51 net.jxta.impl.endpoint.tcp.TcpMessenger closeImpl<br>INFO: Normal close (open 21125ms) of socket to : tcp://193.144.228.57:9765 / 193.144.228.57:9765<br>• End of the first call.<br>• A second call is started.<br>A lot of mesagge ….<br>14-sep-2009 14:23:57 net.jxta.impl.endpoint.tcp.TcpMessenger <init><br>INFO: Creating new TCP Connection to : tcp://193.144.228.57:9765 / 193.144.228.57:9765<br>14-sep-2009 14:23:58 net.jxta.impl.endpoint.tcp.TcpTransport getMessenger<br>ADVERTENCIA: Could not get messenger for tcp://193.144.228.57:9765 : Connection<br>refused: connect<br><br>• The second call fails to connect because the port was previously closed. We reviewed that the second time the service is invoked the port number is the same that the use in the first call, and that it was already closed by the server.</td></tr>
<tr><td>Flypeer</td><td>R0.S.0006</td><td>• A node is trying to use a service calling it with sequence methods.<br>• The service is started and try to acces to the parameters.keySet() trough the method serviceEvent. The service receives the Keys as null when it should be subTransactionId although the serviceParameter is contained.</td></tr>
<tr><td>Flypeer</td><td>R0.S.0007</td><td>• The composition of service calls is performed by the creation of a single XML file. This file should contain a call for each of the invoked services.</td></tr>
</table>

| | |
|---|---|
| | • If we, instead of creating a XML file, build a service composition from source code and we call 3 times to the services:<br>o The first call works right.<br>o The rest of the calls fails.<br>• If a timer is introduced between the calls waiting for the requests of the service invocations, no error is received. |

**Table 27.- Evaluation of the Flypeer 0.3.1 version.**

## *A.II.2.5 Errors Summary.*

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R0.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R0.D0001 | 30/09/2009 | |
| | R0.D0002 | 30/09/2009 | |
| | R0.D0003 | 30/09/2009 | |
| | R0.S0001 | 30/09/2009 | |
| | R0.S0001 | 30/09/2009 | |
| | R0.S0001 | 30/09/2009 | |
| | R0.S0001 | 30/09/2009 | |

# Appendix III. R2 Details of Tests Performed.

## A.III.1 Environment Description.

**Environment Description.**

| | |
|---|---|
| **Date of the Report:** | 18/12/2009 |
| **Module/Platform Version:** | Flypeer 0.5. (Released on 20091218) |
| **Documentation:** | There are 3 manual versions<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |

## A.III.2 Test R2.0003

### A.III.2.2 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.
We considered next network topology in order to run the test:

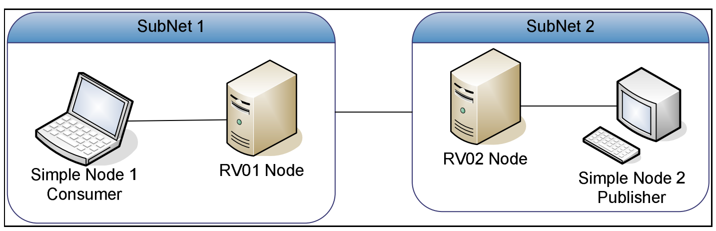



**Figure 28.- Network Topology**

We next describe the previous diagram shortly:

- 2 Rendezvous nodes are deployed (RV01 and RV02) under different subnets.
  - Error in documentation: there is not any indication that in the beginning of a JXTA network each of the RV nodes has to refer/be "linked" each other.
- A simple node (Simple Node 1) is deployed into the Subnet1 and it is "linked" to RV01 node. This node is going to consume the "*Helloword"* service available in the Simple Node 2.
- Another simple node (Simple Node 2) is deployed in the other subnet an linked to the local Rendezvous (RV02). This node published a service "*HelloWorld*".

### A.III.2.3 Documentation Errors.

| Document | Error id | Description |
|---|---|---|
| D3 | R2.D.0001 | There are not a documentation to put in NetBeans the file rendevous_ip.properties and seeds.properties |
| D3 | R2.D.0002 | To executed the example, we copy form web the codesource and fixed import libraries. There are not downloaders binary to execute the example. |

| | | |
|---|---|---|
| **D3** | R2.D.0003 | There're error in Wiki to deployment the service |
| **D3** | R2.D.0004 | There are not instructions to execute in SO Microsoft Windows. |
| **D3** | R2.D.0005 | There are not instructions to deploy an applet and sign. If you don't sign a applet, the applet does not work |
| **D3** | R2.D.0006 | Lost instructions that specific inside the file service in format jar it is need inset the file WSDL |
| **D1** | R2.D.0007 | Error in name of label seed.<br>Said "seed_1" and "seed_2" and it must said "seeds_1" and "seeds_2" |

## A.III.2.4 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer (HelloWord)** | R2.S.0008 | The Peer are execute in applet inside a Web browser don't find rendevous's service.<br><br>Applet log:<br><br>*INFO: Loaded module : General Purpose Peer Group Implementation (net.jxta.impl.peergroup.StdPeerGroup)*<br>*18-dic-2009 14:19:31 net.jxta.impl.peergroup.GenericPeerGroup getInterface*<br>*INFO: [urn:jxta:uuid-D398B66CCB724A079AC39A49A898B4D5A01BE8FBAEC740BDB2B8B925CABF922602] GROUP REF COUNT INCREMENTED TO: 1 by net.jxta.impl.peergroup.PeerGroupInterface.newGroup(PeerGroupInterface.java:315)*<br>*18-dic-2009 14:19:31 br.org.ipti.flypeer.handler.PeerGroupHandler connectToPeerGroup*<br>*INFO: math Peer Group Found ...*<br>*18-dic-2009 14:19:31 br.org.ipti.flypeer.handler.PeerGroupHandler connectToPeerGroup*<br>**INFO: math Peer Group Joined ...**<br>*18-dic-2009 14:19:42 net.jxta.impl.pipe.NonBlockingWireOutputPipe <init>*<br>*INFO: Constructing for urn:jxta:uuid-59616261646162614E5047205032503350656572566694577B575A9642D41303104*<br>*18-dic-2009 14:19:42 net.jxta.impl.pipe.NonBlockingWireOutputPipe close*<br>*INFO: Closing queue for urn:jxta:uuid-59616261646162614E5047205032503350656572566694577B575A9642D41303104*<br>*Exception in thread "Thread-26" java.lang.UnsupportedOperationException: Not supported yet.*<br>*at* **MessageProcessor$SumListener.processFail***(MessageProcessor.java:73)*<br>*at br.org.ipti.flypeer.trans.TransactionInitiator$1.run(TransactionInitiator.java:104)*<br>*at java.lang.Thread.run(Unknown Source)*<br>*18-dic-2009 14:20:31 net.jxta.impl.pipe.NonBlockingWireOutputPipe <init>*<br><br>Rdv log<br><br>18-dic-2009 14:17:11 net.jxta.impl.discovery.DiscoveryServiceImpl startApp<br>INFO: Discovery service started<br>18-dic-2009 14:17:11 net.jxta.impl.cm.SrdiIndex <init><br>INFO: [urn:jxta:uuid-D467CBA3A4BF43A0AD8E96A4FC766DE3A01BE8FBAEC740BDB2B8B925CAB<br>F922602 "math"[0] / urn:jxta:uuid-A01BE8FBAEC740BDB2B8B925CABF922659616261646162<br>614E5047205032503302 "FLYPEER_DEFAULT"[2] / urn:jxta:jxta-NetGroup "NetPeerGroup<br>"[2] / urn:jxta:jxta-WorldGroup "World PeerGroup"[1]] : Initialized pipeResolver Srdi<br>18-dic-2009 14:17:11 net.jxta.impl.pipe.NonBlockingWireOutputPipe <init> |

| | |
|---|---|
| | INFO: Constructing for urn:jxta:uuid-<br>59616261646162614E50472050325033DEADBEEFDEA<br>FBABAFEEDBABE0000000F04<br>18-dic-2009 14:17:11 net.jxta.impl.cm.SrdiIndex startGC<br>INFO: [urn:jxta:uuid-<br>D467CBA3A4BF43A0AD8E96A4FC766DE3A01BE8FBAEC740BDB2B8B925C<br>AB<br>F922602 "math"[0] / urn:jxta:uuid-<br>A01BE8FBAEC740BDB2B8B925CABF922659616261646162<br>614E5047205032503302 "FLYPEER_DEFAULT"[2] / urn:jxta:jxta-NetGroup<br>"NetPeerGroup<br>"[2] / urn:jxta:jxta-WorldGroup "World PeerGroup"[1]] : Starting SRDI GC Thread<br>for pipeResolverSrdi<br>18-dic-2009 14:17:11 net.jxta.impl.peergroup.GenericPeerGroup loadModule<br>INFO: Loaded module : General Purpose Peer Group Implementation<br>(net.jxta.impl.p<br>eergroup.StdPeerGroup)<br>18-dic-2009 14:17:11 net.jxta.impl.peergroup.GenericPeerGroup getInterface<br>INFO: [urn:jxta:uuid-<br>D467CBA3A4BF43A0AD8E96A4FC766DE3A01BE8FBAEC740BDB2B8B925C<br>AB<br>F922602] GROUP REF COUNT INCREMENTED TO: 1 by<br>net.jxta.impl.peergroup.PeerGroupInterface.newGroup(PeerGroupInterface.j<br>ava:315)<br>18-dic-2009 14:17:11 br.org.ipti.flypeer.handler.PeerGroupHandler connectToPeerG<br>roup<br>INFO: math Peer Group Found ...<br>18-dic-2009 14:17:11 br.org.ipti.flypeer.handler.PeerGroupHandler connectToPeerG<br>roup<br>INFO: math Peer Group Joined ...<br>18-dic-2009 14:17:11 net.jxta.impl.pipe.InputPipeImpl <init><br>INFO: Creating InputPipe for urn:jxta:uuid-<br>D467CBA3A4BF43A0AD8E96A4FC766DE3CD5E6<br>60A42764F9D8B53C2D6B587996804 of type JxtaUnicast with queue<br>18-dic-2009 14:17:11 br.org.ipti.flypeer.main.ServiceInitializer deployService<br>**INFO: Service flypeer://SUM has been deployed in the group math**<br>The peer is started<br>18-dic-2009 14:17:19 net.jxta.impl.rendezvous.rpv.PeerView seed<br>INFO: New Seeding... |

**Table 29.- Evaluation of the Flypeer 0.5.0 version.**

### A.III.2.5 Errors Summary.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R2.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R2.D0001 | 18/12/2009 | No |
| | R2.D0002 | 18/12/2009 | No |
| | R2.D0003 | 18/12/2009 | Yes |
| | R2.D0004 | 18/12/2009 | No |
| | R2.D0005 | 18/12/2009 | Yes |
| | R2.D0006 | 18/12/2009 | No |
| | R2.D0007 | 18/12/2009 | No |
| | R2.S0008 | 18/12/2009 | No |

# Appendix IV. R3 Details of Tests Performed.

## *A.IV.1 Environment Description.*

| Environment Description. | |
|---|---|
| **Date of the Report:** | 12/01/2010 |
| **Module/Platform Version:** | Flypeer 0.6.0 (Released on 20100112) |
| **Documentation:** | There are 3 manual versión<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show (D4) |

## *A.IV.2 Test R3.0001*

### A.IV.2.2 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.
We considered next network topology in order to run the test:

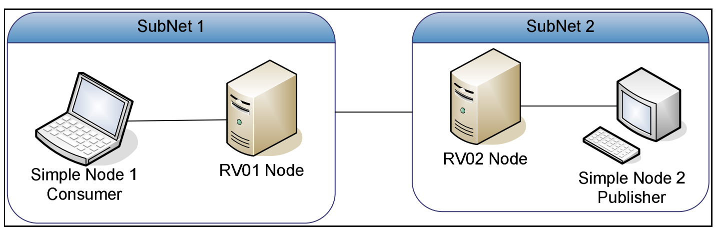



**Figure 30.- Network Topology**

We next describe the previous diagram shortly:
First, we are downloader the examples from repository mercurial. the services Agency, hotel and flightbooking.

- 2 Rendezvous nodes are deployed (RV01 and RV02) under different subnets.
  - Error in documentation: there is not any indication that in the beginning of a JXTA network each of the RV nodes has to refer/be "linked" each other.
- A simple node (RDV Node 1) is deployed into the Subnet1 and it is "linked" to RV01 node. This node is going to consume the "agency" service available in the Simple Node 2. The environment of execution are the Netbeans version 6.5 with 193.144.228.54

- Another simple node (Simple Node 2) is deployed in the other subnet an linked to the local Rendezvous (RV02). This node published a service "flightBooking" and "hotel" service with 193.144.231.64

### A.IV.2.3 Documentation Errors.

| Document | Error id | Description |
|---|---|---|
| **D1, D2,D4** | R3.D.0001 | There are not a documentation to put in NetBeans the file rendevous_ip.properties and seeds.properties |
| **All** | R3.D.0002 | There is not any suggestion about creations project in NetBeans. First, you must created a project in NetBeans and second, you must create another project one for each service to compiling project. |

## A.IV.2.4 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer** | R3.S.0003 | Executen the Flypeer-0.6.jar and thow a exception.<br><br>the log when the service call work right:<br><br>INFO: Started JXTA Network!<br>Exception in thread "Thread-1" java.lang.NoClassDefFoundError: org/tssg/opaals/jxta/authn/AuthenticationType<br>at br.org.ipti.flypeer.main.PeerStarter.run(PeerStarter.java:70)<br>at java.lang.Thread.run(Thread.java:619)<br>Caused by: java.lang.ClassNotFoundException: org.tssg.opaals.jxta.authn.AuthenticationType<br><br>Fix error. |

**Table 31.- Evaluation of the Flypeer 0.6.0 version.**

## A.IV.2.5 Errors Summary.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R2.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R3.D0001 | 12/01/2010 | No |
| | R3.D0002 | 12/01/2010 | No |
| | R3.S0003 | 12/01/2010 | Yes |

## *A.IV.3 Environment Description.*

| Environment Description. | |
|---|---|
| **Date of the Report:** | 18/01/2010 |
| **Module/Platform Version:** | Flypeer 0.6.1 (Released on 20010114) |
| **Documentation:** | There are 3 manual versions<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show(D4) |

## *A.IV.4 Test R3.0001*

### A.IV.4.1 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.
We considered next network topology in order to run the test:

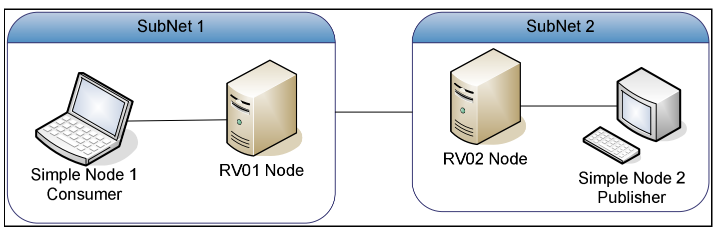
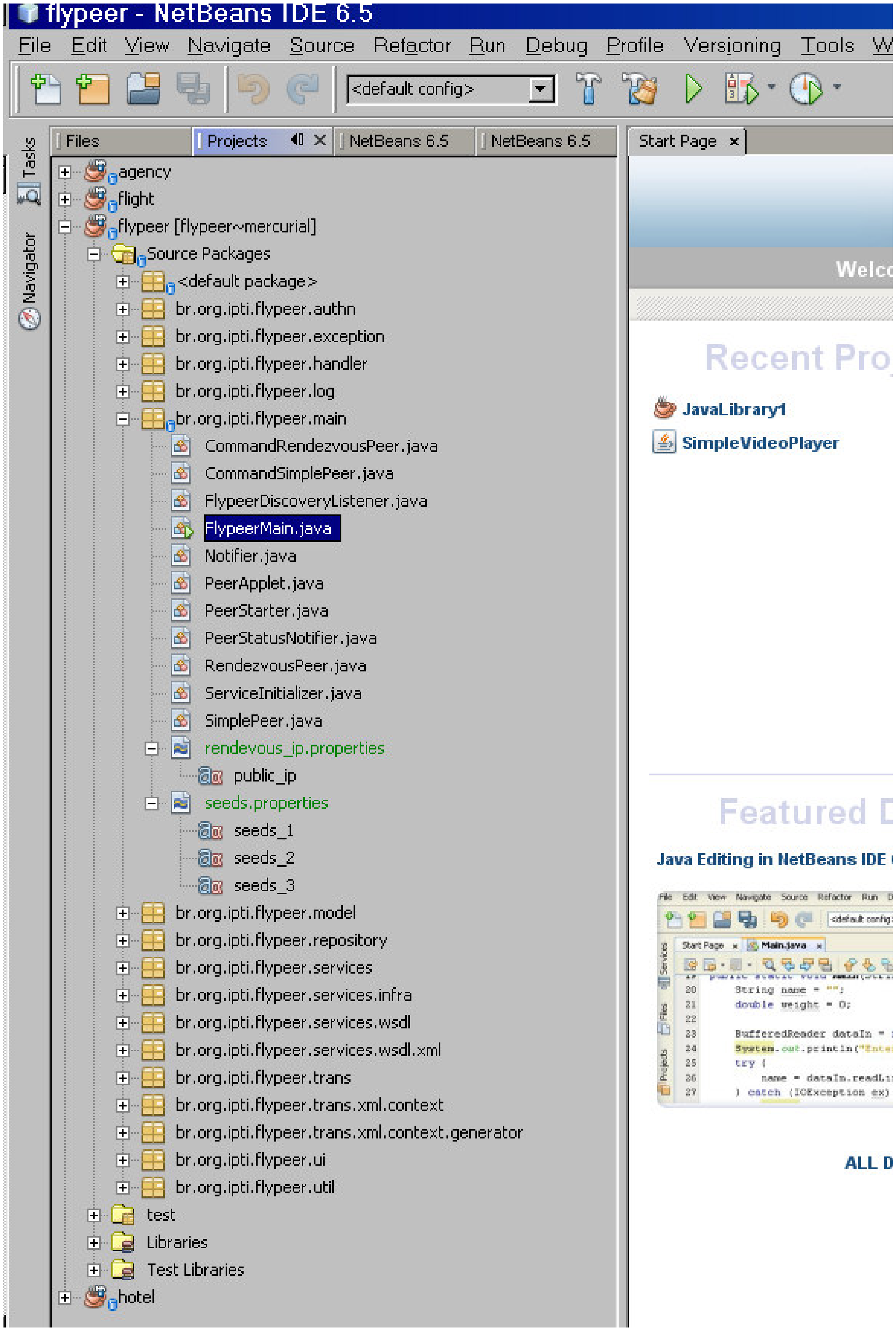


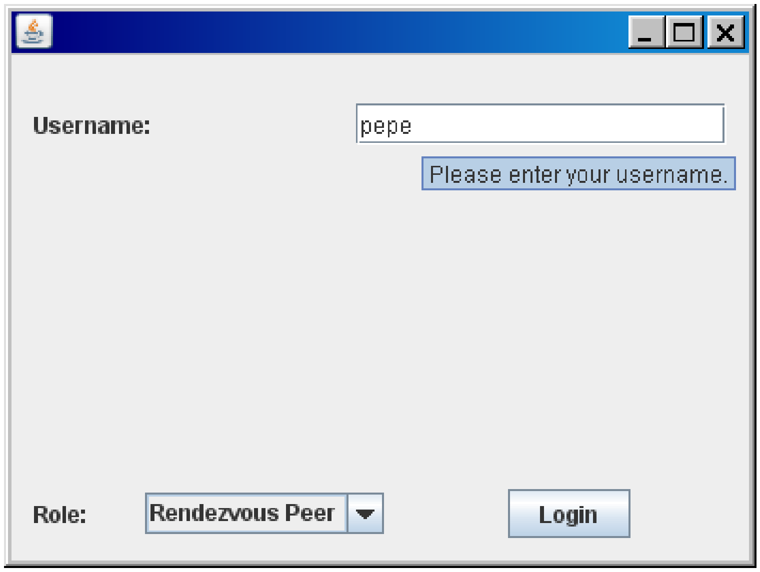

**Figure 32.- Network Topology**

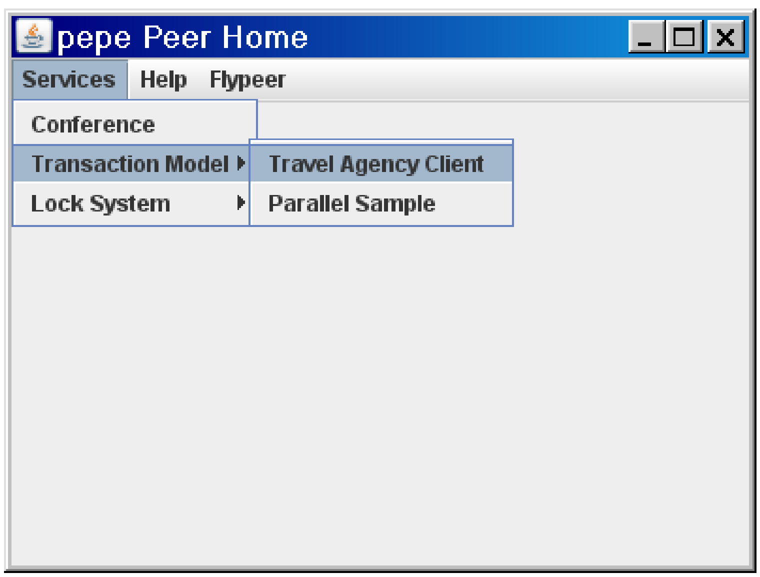
We next describe the previous diagram shortly:
First, we are downloader the examples from repository mercurial. the services Agency, hotel and flightbooking.

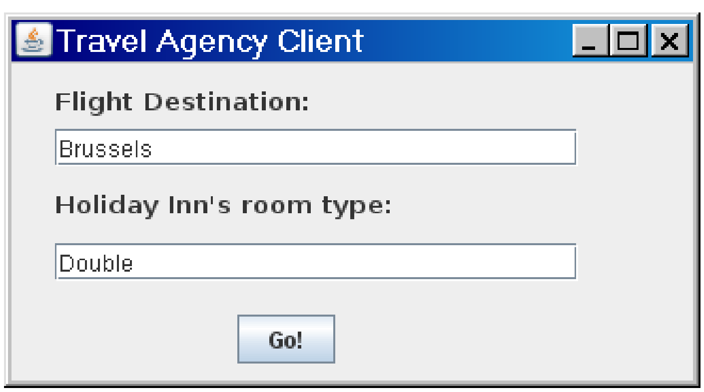
- 2 Rendezvous nodes are deployed (RV01 and RV02) under different subnets.
  - Error in documentation: there is not any indication that in the beginning of a JXTA network each of the RV nodes has to refer/be "linked" each other.
- A simple node (RDV Node 1) is deployed into the Subnet1 and it is "linked" to RV01 node. This node is going to consume the "agency" service available in the Simple Node 2. The environment of execution are the Netbeans version 6.5 with 193.144.228.54

- Another simple node (Simple Node 2) is deployed in the other subnet an linked to the local Rendezvous (RV02). This node published a service "flightBooking" and "hotel" service with 193.144.231.64


flypeer - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools
<default config>
Files
Projects
NetBeans 6.5
NetBeans 6.5
Start Page
Tasks
Navigator
agency
flight
flypeer [flypeer~mercurial]
Source Packages
<default package>
br.org.ipti.flypeer.authn
br.org.ipti.flypeer.exception
br.org.ipti.flypeer.handler
br.org.ipti.flypeer.log
br.org.ipti.flypeer.main
CommandRendezvousPeer.java
CommandSimplePeer.java
FlypeerDiscoveryListener.java
FlypeerMain.java
Notifier.java
PeerApplet.java
PeerStarter.java
PeerStatusNotifier.java
RendezvousPeer.java
ServiceInitializer.java
SimplePeer.java
rendevous_ip.properties
public_ip
seeds.properties
seeds_1
seeds_2
seeds_3
br.org.ipti.flypeer.model
br.org.ipti.flypeer.repository
br.org.ipti.flypeer.services
br.org.ipti.flypeer.services.infra
br.org.ipti.flypeer.services.wsdl
br.org.ipti.flypeer.services.wsdl.xml
br.org.ipti.flypeer.trans
br.org.ipti.flypeer.trans.xml.context
br.org.ipti.flypeer.trans.xml.context.generator
br.org.ipti.flypeer.ui
br.org.ipti.flypeer.util
test
Libraries
Test Libraries
hotel
Recent Pro
JavaLibrary1
SimpleVideoPlayer
Featured
Java Editing in NetBeans IDE
ALL D


Username:
pepe
Please enter your username.
Role:
Rendezvous Peer
Login


pepe Peer Home
Services
Help
Flypeer
Conference
Transaction Model
Lock System
Travel Agency Client
Parallel Sample


Travel Agency Client
Flight Destination:
Brussels
Holiday Inn's room type:
Double
Go!

## A.IV.4.2 Documentation Errors.

| Document | Error id | Description |
|---|---|---|
| **D1, D2** | R3.D.0004 | There are not a documentation to put in NetBeans the file rendevous_ip.properties and seeds.properties |
| **All** | R3.D.0005 | There is not any suggestion about creations project in NetBeans. First, you must created a project in NetBeans and second, you must create another project one for each service to compiling project. |

## A.IV.4.3 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer** | R3.S.0006 | We don't know the configuration environment. Is it need open any port to internet to work?. We have opened the HTTP port to download the http://ipti.org.br/flypeer/TransactionContext.xsd but it does not work.<br><br>The next are some lines recovered from the log. The next is the information provided by the log when the service call work right:<br><br>br.org.ipti.flypeer.exception.IllegalTransactionContextException: Found errors:<br>schema_reference.4: Failed to read schema document 'http://ipti.org.br/flypeer/TransactionContext.xsd', because 1) could not find the document; 2) the document could not be read; 3) the root element of the document is not <xsd:schema>.<br>cvc-elt.1: Cannot find the declaration of element 'TransactionTree'.<br>at br.org.ipti.flypeer.util.TransactionContextParser.load(TransactionContextParser.java:87)<br>at br.org.ipti.flypeer.trans.TransactionInitiator.startTransaction(TransactionInitiator.java:76)<br>at br.org.ipti.agency.service.client.TravelAgencyClient.bookTravel(TravelAgencyClient.java:88)<br>at br.org.ipti.agency.service.ui.TravelAgencyUI.jButton1ActionPerformed(TravelAgencyUI.java:118)<br>at br.org.ipti.agency.service.ui.TravelAgencyUI.access$000(TravelAgencyUI.java:31)<br>at br.org.ipti.agency.service.ui.TravelAgencyUI$1.actionPerformed(TravelAgencyUI.java:77)<br>…<br><br>…<br><br>..<br><br>java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:174)<br>at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:169)<br>at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:161)<br>at java.awt.EventDispatchThread.run(EventDispatchThread.java:122)<br>Exceptions: /n<br>{null=br.org.ipti.flypeer.exception.IllegalTransactionContextException: Found errors:<br>schema_reference.4: Failed to read schema document **'http://ipti.org.br/flypeer/TransactionContext.xsd', because 1) could not find the document; 2) the document could not be read; 3) the root** |

| | |
|---|---|
| | **element of the document is not <xsd:schema>.**<br>cvc-elt.1: Cannot find the declaration of element 'TransactionTree'.}<br>Rollback is succesfull!<br>message "Brussels, Double" sent to the ServiceServer<br>15-ene-2010 14:31:44 net.jxta.impl.rendezvous.rpv.PeerView seed<br><br>And the other node print in log this<br><br>F922602] GROUP REF COUNT INCREMENTED TO: 1 by<br><br>net.jxta.impl.peergroup.PeerGroupInterface.newGroup(PeerGroupInterface.java:315)<br>15-ene-2010 14:31:23 net.jxta.impl.pipe.InputPipeImpl <init><br>INFO: Creating InputPipe for urn:jxta:uuid-BB0830B51A204B868AE5C9E4F45E59C2F7E08D812EB541D7AC86A0255B6618C704 of type JxtaUnicast with queue<br>15-ene-2010 14:31:23 br.org.ipti.flypeer.main.ServiceInitializer deployService<br>INFO: **Service flypeer://BA-FLIGHT-BOOKING has been deployed in the group airline**<br>s<br>15-ene-2010 14:31:23 net.jxta.impl.pipe.InputPipeImpl <init><br>INFO: Creating InputPipe for urn:jxta:uuid-A01BE8FBAEC740BDB2B8B925CABF92263F45E30042AA461384CE9B10B7BFA5F504 of type JxtaUnicast with queue<br>15-ene-2010 14:31:23 br.org.ipti.flypeer.main.ServiceInitializer deployService<br>INFO: **Service flypeer://LUFTHANSA-FLIGHT-BOOKING has been deployed in the group**<br>FLYPEER_DEFAULT<br>The peer is started<br>15-ene-2010 14:31:51 net.jxta.impl.rendezvous.rpv.PeerView seed<br>INFO: New Seeding...<br>15-ene-2010 14:31:56 net.jxta.impl.pipe.NonBlockingWireOutputPipe <init><br>INFO: Constructing for urn:jxta:uuid-5961626164616261 4E504720503250335065657256694577B575A9642D41303104<br>**java.net.SocketTimeoutException: Timeout reached**<br>at net.jxta.socket.JxtaServerSocket.accept(JxtaServerSocket.java:313)<br>at<br>org.tssg.opaals.jxta.JXTAPeerSocketMonitor.run(JXTAPeerSocketMonitor.java:49)<br>15-ene-2010 14:32:19 net.jxta.impl.pipe.NonBlockingWireOutputPipe <init><br>INFO: Constructing for urn:jxta:uuid-59616261646162614E50472050325033506565725669 4577B575A9642D41303104<br>15-ene-2010 14:32:19 net.jxta.impl.pipe.NonBlockingWireOutputPipe close<br>INFO: Closing queue for urn:jxta:uuid-59616261646162614E5047205032503350656572 56 |

**Table 33.- Evaluation of the Flypeer 0.6.1 version.**

## A.IV.4.4 Errors Summary.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R3.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R3.D0004 | 18/01/2010 | No |
| | R3.D0005 | 18/01/2010 | No |
| | R3.S0006 | 18/01/2010 | Yes |

# Appendix V. R4 Details of Tests Performed.

## A.V.1 Environment Description.

| Environment Description. | |
|---|---|
| **Date of the Report:** | 26/02/2010 |
| **Module/Platform Version:** | Flypeer 0.7.RC1 (Released on 20100211) |
| **Documentation:** | There are 3 manual versión<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Service sum** | We received by email from deniswsrosa@gmail.com a service call sum at 03/02/2010 |
| **Others:** | Mercurial source code repository http://kenai.com/projects/flypeer/sources/mercurial/show(D4) |

## A.V.2 Test R4.0001

### A.V.2.1 Test Definition.

We test the **service sum.** The module Sum has two file: a service client and service server. This service is executed in local and remote configuration. It's simple, we only had execute 2 files type JAR.

Local configured: We try executed in two different configuration machines.

- One machine with 2 Interface Network
  - One Service client and one service server
  - Two service client and one service server
- Other machine with one Interface Network.
  - One Service client and one service server
  - Two service client and one service server

Network configuration:

- We try executed two computer

### A. V.2.2 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer** | R4.S.0001 | All test in computer with 2 Network Interface doesn't work |
| | R4.S.0002 | The test with 2 clients doesn't work<br>Talking with Denis, he said that only work with one client. |
| | R4.S.0003 | The Network configuration doesn't work |

**Table 34.- Evaluation of the Sum service with Flypeer 0.7.RC1 version.**

We concluded that a machine with 2 or more Interface Network does not work.

### *A. V.3 Test R4.0002*

## A. V.3.1 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by same node.
We considered no network topology, only local executed

We next describe the previous diagram shortly:
First, we are downloader the examples from repository mercurial. the services Agency, hotel and flightbooking.

- 1 Rendezvous nodes are deployed under local machine.
- 2 simple node with 1 service each node(edge Node 2) is deployed into local machine. This node published a service "flightBooking" and "hotel" service. This node is going to consume the "agency" service available in the RDV node. The environment of execution are the Netbeans version 6.5 in local machine

## A. V.3.2 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer** | | No errors execute in local node |

**Table 35.- Evaluation of the Flypeer 0.7.RC1 version.**

### *A. V.4 Test R4.0003*

## A. V.4.1 Test Definition.

We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node.
We considered next network topology in order to run the test:

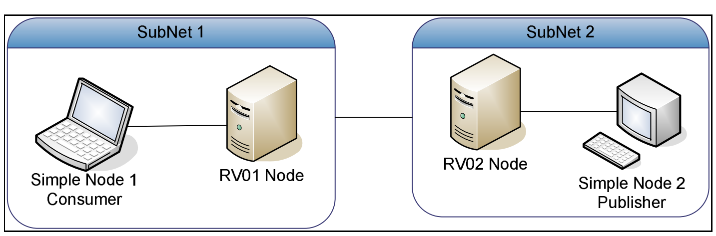



**Figure 36.- Network Topology**

We next describe the previous diagram shortly:

First, we are downloader the examples from repository mercurial. the services Agency, hotel and flightbooking.

- 2 Rendezvous nodes are deployed (RV01 and RV02) under different subnets.
    - Error in documentation: there is not any indication that in the beginning of a JXTA network each of the RV nodes has to refer/be "linked" each other.
- A simple node (RDV Node 1) is deployed into the Subnet1 and it is "linked" to RV01 node. This node is going to consume the "agency" service available in the Simple Node 2. The environment of execution are the Netbeans version 6.5 with 193.144.228.54
- Another simple node (Simple Node 2) is deployed in the other subnet an linked to the local Rendezvous (RV02). This node published a service "flightBooking" and "hotel" service with 193.144.231.64

## A. V.4.2 Software Errors.

| Module | Error id | Description |
|---|---|---|
| **Flypeer** | R4.S.0004 | The Rendezvous are connected between distinct node. This nodes are published in subnet distinct. Next, published the service but don't find the services.<br><br>This is the node log :<br><br>message "Brussels, Double" sent to the ServiceServer<br><br>Exceptions: /n {null=br.org.ipti.flypeer.exception.PeerGroupNotFoundException: Could not find the peer group hotel}<br><br>Rollback is succesfull!<br><br>23-feb-2010 16:14:06 br.org.ipti.flypeer.trans.TransactionInitiator$1 run<br><br>ADVERTENCIA: An error ocurred while starting a new transaction br.org.ipti.flypeer.exception.PeerGroupNotFoundException**: Could not find the peer group hotel**<br><br>at br.org.ipti.flypeer.handler.PeerGroupHandler.connectTo(PeerGroupHandl<br><br>er.java:389)<br><br>at br.org.ipti.flypeer.handler.ServiceHandler.findRemoteServicesByName(S<br><br>erviceHandler.java:70)<br><br>at br.org.ipti.flypeer.main.PreTransactionProcessor.prepare(PreTransacti |

| | |
|---|---|
| | onProcessor.java:75)<br><br>at<br>br.org.ipti.flypeer.main.PreTransactionProcessor.prepareInitiator(Pre<br><br>TransactionProcessor.java:167)<br><br>at br.org.ipti.flypeer.trans.TransactionInitiator$1.run(TransactionIniti<br><br>ator.java:95)<br><br>at java.lang.Thread.run(Unknown Source)<br><br>message "Brussels, Double" sent to the ServiceServer<br><br>23-feb-2010 16:14:29 net.jxta.impl.rendezvous.rpv.PeerView seed<br><br>INFO: New Seeding... |

**Table 37.- Evaluation of the Flypeer 0.7.RC1 version.**

# A. V.2.5 Errors Summary.

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R3.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R4.S0001 | 26/02/2010 | yes |
| | R4.S0002 | 26/02/2010 | yes |
| | R4.S0003 | 26/02/2010 | Yes |
| | R4.S0004 | 26/02/2010 | yes |

# Appendix VI. R5 Details of Tests Performed.

## *A.VI.1 Environment Description.*

| **Environment Description.** | |
|---|---|
| **Date of the Report:** | 20/04/2010 |
| **Module/Platform Version:** | Flypeer 0.8. (Released on 20100407) |
| **Documentation:** | There are 3 manual versión<br>• The flypeer http://kenai.com/projects/flypeer/pages/UserGuide (D1)<br>• The one sent by Thomas Kurz (D2)<br>  • A more complete review of the former.<br>• http://kenai.com/projects/flypeer/pages/HelloWorld (D3)<br>  • The more complete manual as it includes references to the source code to consume a service. |
| **Others:** | Mercurial source code repository<br>http://kenai.com/projects/flypeer/sources/mercurial/show(D4) |

## *A. VI.2 Test R5.0001*

### A. *VI*.2.1 Test Definition.

We implemented only the test suit case RX.R001 defined in this document. We don't created more test because.

✓ The test RX R001:

☒ **Test 01** *Service composer, coordinate***:**

The development environment test were generated in 4 computers. 3 of them are deployed in service and other services are consumed. In each of the nodes where services are deployed have created three services distinct.

*Steep 1:*

Services deployed on each node. In total nine different services are deployed.

*Seep 2:*

Once the services have been deployed and are actively trying to consume.

*Seep 3:*

Seek and consume services.

*Result 1:*

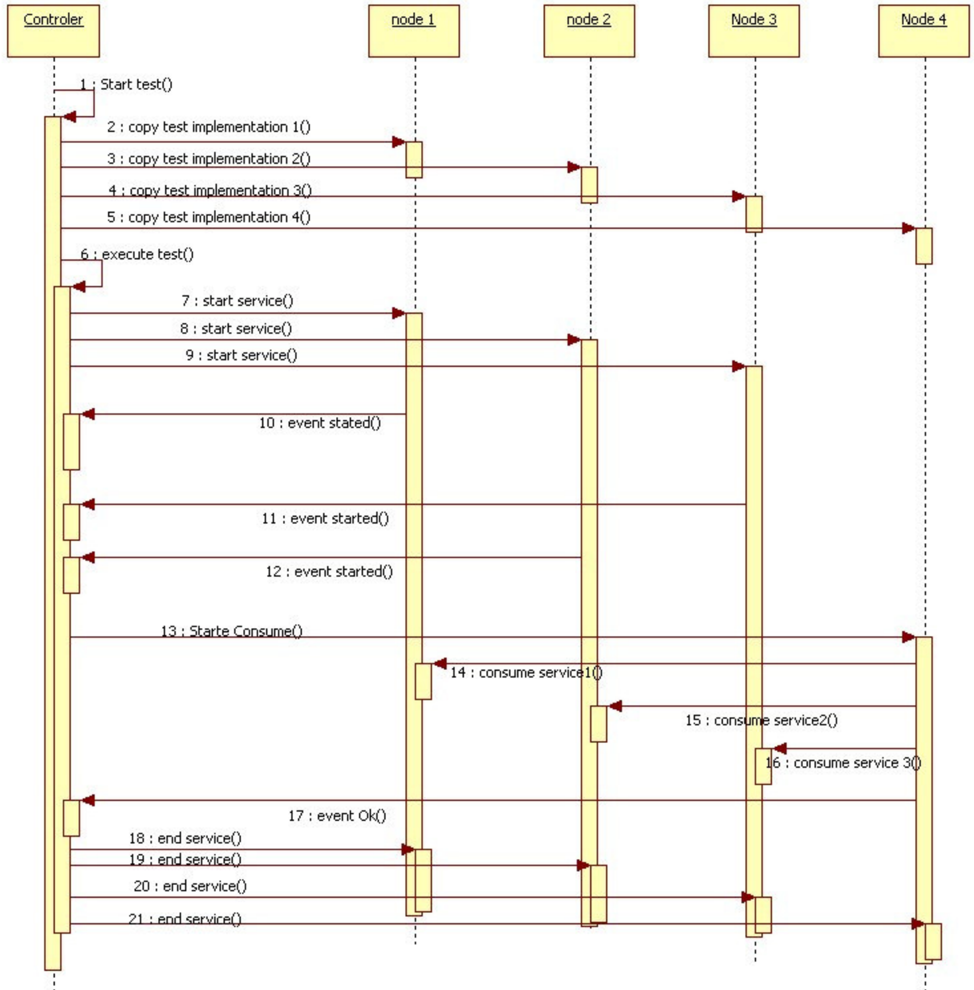
In the steep 2, the service doesn't deploy. You receive an error JXTA

Actions 1: We send e-mail with errors to developer and the developer senf a new release, try again.

*Result 2:*

In the steep 3, the service doesn't find.

Actions 2: We send e-mail with errors to developer and the developers send again a new release. The test doesn't execute.




- ✓ **Test 02** Long-running transaction:
  For this test use a 3 computer or nodes. Establishing a service, that is very slow. That velocity is depending query. A node sends a slow query and other node send a fast query.

  Steep 1: Publish 1 service. And the consumers send query, the fast first and the slow second.

  Result: The slow query block the service. The consume service is synchronous. The fast query send

  Actions: Anything

☒ **Test 03**: Loosely-coupled:

You want to prove that the implementation of the services is not coupled to the network configuration.

Step1: A service is consumed every minute (node 1). The service is consumed, is published in another (node 2).

Step 2: When it is taken for this service and is published in another different node (node 3) the same service. We try to prove if he can discover and consume the new service without restarting (node 1).

Result: The system doesn't find the new service.

Actions: if restart the networks, the new service are finding.

☒ The test case RX.R002 The infrastructure must manage service deployment.

The Javadoc documentation can be found the method *peerStarted*() but you need to implement other methods to indicate the status of deployed services, and the ability to programmatically service manage, not only status of the peer.

In the framework, publishing a service can not be programmatically. The services are deployed through adding them as java libraries. The framework have not dynamic publish.

☒ The test case RX.R003 The system must manage service searching and service consuming manage and provisioning.

Although the class *CommandSimplePeer* is more or less correctly documented, it is not clear the next method:

```
public void getListofServices(java.lang.String service,
                              java.lang.String peerGroup,
                              int maxThreshhold,
                              SearchStatusNotifier notif,
                              long timeout)
   Returns a list of services matching the given criteria.

   Parameters:
      service - name of the service to look for
      peerGroup - peer group to look into for the services
      maxThreshhold - max number of results
      notif - object to be notified when the search results are ready
      timeout - max time the caller is willing to wait for the search to complete
```

1. In this method there are some questions. Why the *SimplePeer* don't implement the method *getListofService* and however

*CommandSimplePeer* implement this method. Are there some diference?

2. As a prerequisite to find a search service is required within a group. What exactly is a group? What is the purpose of a group? How does it manage groups? Who creates them? How does it find them? If you only want to search for a service, such a search is performed without groups?

3. Problems entailed in the search method. Particularized to the case of hotels in a city. How is the search of the reservation? If you wish to reserve a hotel in Madrid. "Hotel Madrid" service provider of hotel in Madrid, and a client is seeking booking service.
    a. Option:
    *Point of view from the service provider*: The service provider deploys 2 hotel services. One reservation service and one localization service hotel in the city.
    *Client or User*: the user wants to find a hotel in Madrid. Search hotel group, to service location and ask all hotels hotel location. Once found the Hotels in Madrid seek the service of this hotel for booking.
    Scalability: It is not possible to talk to all the services of all hotels in the world. Great amount of time and requests.

    b. Option:
    *Point of view from the service provider:* The service provider deploys 2 hotel services. One reservation and one for location hotel in the city genus is a group of "hotel madrid"

    c. Option:
    *Point of view from the service provide:* The service provider deploys a service in groups: "hotel", "hoteles" (in Spanish), "hostel", "hostels" and "Madrid"
    *User:* User search "hotels" and look in the "Madrid" and it makes an intersection with the results. After that, the user looking the hotel service name with the words "reservation", "booking" or "reservation" (in French).

    *Problems:*
    How to link services different of same hotel? Each service is deployed separately without relation between the services of the same hotel. Scalability: It is not possible to talk to all the services of all hotels in the world. Great amount of time and requests.
    No documentation to tell us how we should make the deployment of services to make a correct search.

- ✓ The test case RX.R004, The infrastructure must allow for service composition.
  Each service has two types of class composition:
  - ✓ Static service composition:
    a. The user should know service description: it's correct because there is a document *WSDL* that description the parameters.
    b. The user should know the semantic meaning. It's correct if the deployment publish a file with this information. In The WSDL file there're points to description.

  - ☒ Dynamic service composition:
    c. All operation performed by the user through code in static mode, must be obtained through the system itself. This is true, because there are *br.org.ipti.flypeer.services.wsdl.xml*. The framework must be obtained through a set of rules. This is false.

  - ✓ Test 1: Does the system provide semantic meaning parameter?
    Supporter by WSDL file

  - ☒ Test 2: Does the system provide conversion units?
    No supporter.

  - ☒ Test 3: Does the system provide adapters?
    No provider adapter form WS to flypeer service.

  - ✓ Test 4: Check if a simple static service composite can be created.
    Yes, it's creating a simple static service.

  - ☒ Test 5: Validate if a simple dynamic service composite can be created.
    No, the rules are not support.

  - ✓ Test 6: Check if a complex static service composite can be created.

  - ☒ Test 7: Check if a complex dynamic service composite can be created.
    No, the rules are not support.

- ☒ The test case RX.R005, the system must manager security.

  - ☒ Protection of data to avoid unauthorized access.
  - ☒ Systems won't allow read or modify data to unauthorized persons.

- ☒ Systems will provide access to authorized persons.
- ☒ The system should protect itself from accidental or malicious accesses.

There is no unauthorized access control. When a service is deployed everybody can used.

- ✓ The test case RX.R006 The infrastructure must manager identity, confidentiality

  The framework used library "The Bouncy Castle Crypto Packaged", this is a Java implementation of cryptographic algorithms. To send user data, are using the Digital Signature Algorithm (DSA). Public key cryptography is a fundamental and widely used technology around the world. It is the approach which is employed by many cryptographic algorithms and cryptosystems. It underlies such Internet standards as Transport Layer Security (TLS) (successor to SSL), PGP, and GPG.
  Some information data, belong the user (login and password) is send encrypting.

- ✓ The test RX.R007, the infrastructure must make bootstrapping connection easier.
  The framework can connect to the networks. The frameworks only connected to the networks when all parameters are configured.

- ✓ The test RX.R008, the infrastructure must give support for scalability and performance, there're a lot papers to talking about the good scalability and performance:
  "JXTA, be designed to provide good performance, high scalability, and adaptation to heterogeneous environment"
  "Earlier results indicate that a broad and more detailed performance study is needed. The JXTA community initiated a dedicated sub-project, with a purpose to collect performance and scalability measurements as the platform development progresses" [13]
  The base of the framework is JXTA, which supports architecture P2P. This is only a part of the framework, there're another pieces, which it's necessary to prove. For example, an correct implementation
  A P2P system has a much easier time of it. Since no centralized system monitors everything, there are no problems with system capacity: users of the service monitor nearby services in which they are interested and discard other information.[16]

- ✓ The test RX.R009, the infrastructure should be based on standards. The frameworks use a few standards only in definition of parameters, but the

frameworks haven't standards to in the identity or in methods to services consume the framework send java object, and this are not standard.
The framework is opening to outside the system with web service. This use Extensible Markup Language (XML) messages that follow the Simple Object Access Protocol (SOAP) standard with an XML serialization in conjunction with other web-related standards.

- ✓ The test RX.R010, the infrastructure should run on different software and hardware platforms- portability.

In computing, cross-platform, or multi-platform, is an attribute conferred to computer software or computing methods and concepts that are implemented and inter-operate on multiple computer platforms. Cross-platform software may be divided into two types; one requires individual building or compilation for each platform that it supports, and the other one can be directly run on any platform without special preparation, e.g., software written in an interpreted language or pre-compiled portable *bytecode* for which the interpreters or run-time packages are common or standard components of all platforms.
The infrastructure made in Java and Java support this feature. The heart of the Java Platform is the concept of a "virtual machine" that executes Java *bytecode* programs. The use of *bytecode* as an intermediate language permits Java programs to run on any platform that has a virtual machine available. Although Java programs are platform independent, the code of the Java Virtual Machine (JVM) that execute these programs is not; every supported operating platform has its own JVM.[15]

- ✓ The test RX.R011, the infrastructure should have support for asynchronous communication.

Invocations of services are asynchronous in nature in that the service provider must be capable of accepting requests from the other service without notice. However, sometimes the response to the service request is available on the same thread of execution as the invocation; such operations are often labelled as synchronous. This discussion of asynchronous operations will not focus on the initiation of request messages by clients or the consumption of request messages by service providers; rather, I'll focus on how to handle responses to service requests that are not provided immediately but at a time after the initial request transactions complete. Such asynchronous behaviour is common for services that require complex processing that may take minutes or even days to complete -- when, for example, the service implementation is dependent on batch processing or manual steps requiring human intervention [14].

The asynchronous communication is support for the framework. Only in service consume client, when the service provider don't support asynchronous communication.

- ✓ The test RX.R012, the infrastructure should implement some self-healing, self-optimization or self-protection properties.

  The implementation based on a JXTA middleware, shows that P2P model through its self-organizing characteristic can improve applications performance by providing a maximal autonomy for a minimal service configuration. This approach provides new signalling mechanisms and components for coordinating, managing and controlling, in a dynamic network of peers.

  The infrastructure implements self-discovery with anther nodes.

- ✓ The test RX.R013, the infrastructure should use p2p network architecture decentralization.
  The framework use JXTA technology. JXTA is a set of open protocols that allows any connected device on the network to self-organize and collaborate in a peer-to-peer (P2P) manner without requiring centralization. JXTA lets peers create and join a multitude of application-defined virtual network domains in which any peer can interact with other peers and resources directly, even when some of the peers and resources are behind firewalls and network address translations (NATs) or are on different network transports
  JXTA is a P2P approach for supporting type services in a decentralized architecture. The infrastructure implements an architecture base in p2p.

- ✓ The test RX.R014, the infrastructure must ensure robustness. The infrastructure is robustness, because errors only affect to part of the system stability.
  There's no centralized resource controlling everything, the P2P community is less affected by network failures or denial of service attacks. These problems will still affect individual participants in the P2P community, but the failures they cause will not have a widespread effect on others within the community.[16]

- ☒ The test RX.R015, the infrastructure must ensure reliability. The ability to recover from an error in the system and return to the previous state before the error occurred.
  Found an article on reliability on the JXTA protocol. Due to JXTA is the basis of the architecture of the Framework. It is extremely important, independent of the implementation of the Framework
  Analyzed the JXTA protocols and services have shown several limitations of these protocols regarding the efficiency and reliability of P2P JXTA-based applications. The analyzed protocols include

discovery, peer information, peer resolver, and pipe binding protocols/services, among others. We observed the need for improving the original JXTA protocols, such as pipe services, to ensure reliable communication between peer nodes and the discovery and presence service to increase the performance of the applications. The re-implemented protocols have been validated in practice by deploying a P2P network using nodes of PlanetLab platform and testing each of the extended protocols using this real P2P network."[18]

- ✓ The test RX.R016, the system must provide integration with system that are on different architectures offering interoperability.

  Interoperability is the most important principle of SOA. This can be realized through the use of web services, as one of the key benefits of web services is interoperability, which allows different distributed web services to run on a variety of software platforms and hardware architectures.

  Interoperability and portability start with the standard specifications themselves.

  The Interoperability is important because it is necessary to exchange messages between different platform. The hardware of a machine is abstracted by the operating system that supports that architecture. In turn, the Java Virtual Machine abstracts operating system supports. The Framework has been implemented in Java, the only point in mind is the support of the JVM. This, is implemented for a large number of operating systems.

- ▪ The test RX.R017, the infrastructure should be transactional.

  The conventional definition of a transaction is based on ACID properties: **A**tomicity – either all tasks in a transaction are performed, or none of them are; **C**onsistency – data is in a consistent state when the transaction begins, and when it ends; **I**solation – all operations in a transaction are isolated from operations outside the transaction; **D**urability – upon successful completion, the result of the transaction will persist. However, in advanced distributed applications these properties can present unacceptable limitations and reduce performance, a view also supported in [17].

  Not implemented a classic definition of transactions, which does not apply this requirement.

## A. *VI.*2.2 Software Errors.

| Module | Error id | Description |
|---|---|---|

| | | |
|---|---|---|
| **RX.R001-Test1** | R5.S.0001 | An error in JXTA: Group already instantiated. Don't works |
| **RX.R001-Test2** | R5.S.0002 | The service is implemented in Flypeer, which must answer the call of another sevice, but this, is blocked by a third service. No multi-thread implemented response. The response of service are bloking |
| **RX.R001-Test3** | R5.S.0003 | If a service (Service 1) is published, and this is consumed by other services (Service 2), when re-post the service (Service1) without making any changes, the service is not found at (Service2) until you restart service (Service 2) |

**Table 38.- Evaluation of the Sum service with Flypeer 0.8.RC1 version.**

## *A. VI.2.5 Errors Summary.*

| Test Id | Description | We are going to follow documentation instructions: we are going to try to consume a service which is provided by another node. | |
|---|---|---|---|
| **R3.0001** | **Bug Id** | **Reported on** | **[To Be]Fixed on** |
| | R5.S0001 | 30/04/2009 | No |
| | R5.S0002 | 30/04/2009 | No |
| | R5.S0003 | 30/04/2009 | No |